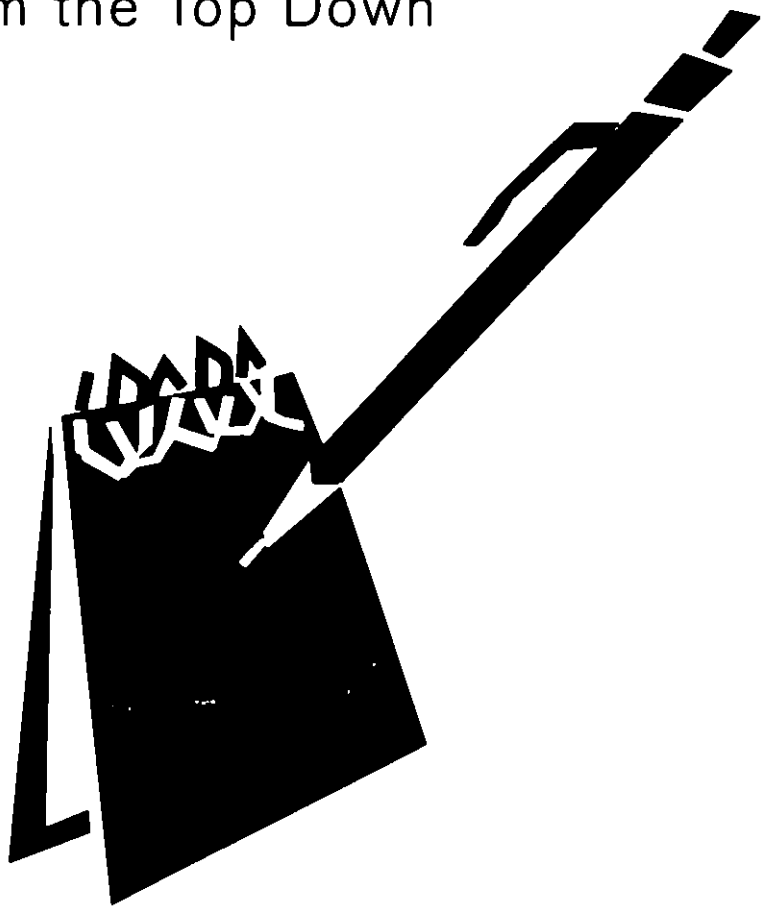
SIGGRAPH 98

20

Real-Time Graphics for Visual Simulation: Advanced Techniques from the Top Down



25th International Conference on Computer Graphics and Interactive Techniques
Exhibition **21 23 July** 1998 Conference **19 24 July** 1998
Orlando Florida USA

course notes

SIGRAPH 98

20

Real-Time Graphics for Visual Simulation Advanced Techniques from the Top Down

Organizer
Elizabeth Smith
Paradigm Simulation Inc

Lecturers
Graham Beasley
Sharon Rose Clay
Silicon Graphics Inc

Dan Brockway
Stephen Gersuk
Elizabeth Smith
Paradigm Simulation Inc

J Bruce Howie
Lockheed Martin Information Systems

Ronald Moore
Evans & Sutherland Computer Corporation

25th International Conference on Computer Graphics and Interactive Techniques
Exhibition **21 23 July** 1998 Conference **19 24 July** 1998
Orlando, Florida USA

course notes

Real-Time Graphics for Visual Simulation
Advanced Techniques from the Top Down

SIGGRAPH 98 Course 20 Notes

Course Organizer
Elizabeth Lee Smith
Paradigm Simulation Inc

Lecturers
Graham Beasley
Silicon Graphics Inc

Dan Brockway
Paradigm Simulation Inc

Sharon Rose Clay
Silicon Graphics Inc

Stephen Gersuk
Paradigm Simulation Inc

J Bruce Howie
Lockheed Martin Information Systems

Ronald G Moore
Evans and Sutherland

Abstract

Many techniques for computer graphics had their beginning in early real time visual simulation (vis sim) projects Algorithms originally developed for flight simulators running on multi million dollar systems have evolved into useful methods for real time interactive applications of all sorts on significantly less expensive computers While the processing power and graphics capabilities of PCs and game systems are steadily increasing so are the capabilities of high end graphics systems This course will explore those interactive visual simulation algorithms and tricks for which high-end hardware is currently required These techniques span all aspects of vis sim application development from visual database construction techniques to multiprocessing considerations for displaying large geographic areas in real time to image distortion correction for dome display If past trends continue these procedures will be adapted for low end use as desk top systems become more capable

Real-Time Graphics for Visual Simulation

Table of Contents

Abstract	i
Table of Contents	ii
Speaker Biographies	iii
Course Syllabus	vi
Introduction and Overview	1 1
History and Requirements of Visual Simulation	2 1
Introduction to Computer Image Generation Architectures	2 22
Multiprocessing and Graphics Pipelines (Run Time Concepts)	
Evans and Sutherland Architectures for Real-Time Display of Large Areas and Big Textures	3 1
Ways to Handle Geometry and Textures for Large Geographic Areas in Real-Time	4 1
Silicon Graphics Architectures for Real-Time Display of Large Areas and Big Textures	5 1
Importance and Requirements of Multispectral Applications	6 1
Real-Time Generation of Multispectral Images	6 8
Visual Database Construction Techniques	
Database Designs	7 1
Large Area Texture and Terrain	8 1
Integrate Feature Modeling	9 1
Performance Management	10 1
Displaying the Results	
Special Displays for Visual Simulation	11 1
Creative Uses of Bitplanes	12 1

Appendices

A1 Cost-Effective Interoperable Distributed Interactive Simulation for Large-Scale Tactical Warfare Simulation
A2 Database Development for Advanced Distributed Simulation
A3 Multiple Image Suppression in a Low-Cost Visual System
A4 Non-Linear Distortion Correction
A5 Real-Time Image Visualization for Sensors

Color Images

Speaker Biographies

Graham Beasley

Silicon Graphics Inc
2011 N Shoreline blvd #525
Mountain View CA 94039
graham@sgi.com

Graham Beasley has been employed by Silicon Graphics (SGI) since 1990 where he has worked in the Advanced Graphics Division His specialties include Real Time simulation visual database networked simulation and helmet mounted displays Before joining SGI Mr Beasley worked at CAE Electronics (the world s largest simulation company) from 1984 1990 in several of the image generator groups Currently Mr Bealey manages an engineering group specializing in Modeling Simulation and Imagery He has published papers and given courses to various professional associations including Image Society Interservice/Industry Training Systems and Education Conference (I/ITSEC) SPIE He earned a Bachelor of Computer Science from Concordia University

Dan Brockway

Paradigm Simulation Inc
14900 Landmark Blvd Suite 400
Dallas TX 75240
db@paradigmsim.com

Dan Brockway manages database development programs at Paradigm Simulation Inc in Dallas Texas He received a Bachelors of Environmental Design from Texas A&M University in 1984 where he studied Computer Graphics and Architecture At Greenhorne & O Mara in Greenbelt Maryland he developed GIS applications and courses to train engineers in the adaptation of Computer Aided Design technology He authored "101 User commands" a compendium of utility software published by OnWord Press 1990 He entered the Training and Simulation field at Cambridge Research in 1991 by writing a GI based image generator for a Ball/MegaTec 944 At SAIC he managed the Advanced Graphics team for the ARPA War Breaker project There he developed the World Reference Model an architecture for managing database production in distributed interactive simulation environments In 1995 he used the World Reference Model as the basis for a course at the IMAGE conference At SAIC he built databases of Korea and Nellis for the F15 Weapons Tactics Trainer Now at Paradigm Simulation his primary focus is development of databases designed to meet rigorous customer requirements He also serves as an advisor to the Synthetic Environment Data Representation and Interchange Specification (SEDRIS) core team

Sharon Rose Clay

Silicon Graphics Inc
2011 N Shoreline Blvd Bldg 8 MS 590
Mountain View CA 94043 1389
src@sgi.com

Sharon Rose Clay is the manager of the IRIS Performer® engineering team in the Advanced Graphics Software department at Silicon Graphics She specializes in performance issues for system implementation and real time graphics applications She was a member of the original Performer design team and before that, was a member of the Graphics Software group where she worked on the development team for the VGX graphics platform Her interests besides a real need for speed include user interfaces plants and fish (simulated and real) She studied using natural language in graphical user interfaces at the University of

California at Santa Cruz where she receive her Masters degree in Computer Science Her Bachelors degree is in Mathematics and Linguistics from the University of California at Berkeley She has taught several courses at past SIGGRAPH conferences dealing with real time issues in graphics

J Bruce Howie

Lockheed Martin Information Systems
12506 Lake Underhill Dr
Orlando FL 32817
howie@escmail orl lmco com

J Bruce Howie is a staff systems engineer with Lockheed Martin Information Systems in Orlando FL He received an BS degree in Physics at the University of Texas at Arlington in 1981 with some graduate work In 1983 he joined Rediffussion Simulation as a Visual Database Engineer where he developed real-time databases for flight and maritime applications In 1995 he joined General Electric Aerospace (now Lockheed Martin Information Systems) Since 1995 he has been involved in the production of real time databases the education of real time database generation to internal and external customers and most recently the systems engineer responsible for the CompuScene TARGET database generation software He is also the project engineer responsible for the LMIS SEDRIS (Synthetic Environment Data Representation and Interchange Specification) effort with DARPA DMSO and STRICOM He has co-authored 17 papers in physics while at university and 2 in visual simulation professionally The most recent paper given at the Fall 1997 Simulation Interoperability Workshop

Stephen Gersuk

Paradigm Simulation Inc
14900 Landmark Blvd Suite 400
Dallas Texas 75240
shg@paradigmsim com

Stephen Gersuk is Vice President of new business development for Paradigm Simulation He has fifteen years experience in visual system design for simulation and training applications and a special interest in quantitative analysis of panoramic display systems He holds two patents for optimal surfaces for real image projection and served as Chair for two years for the IMAGE Society s Special Interest Group for Displays Prior to joining Paradigm Mr Gersuk spent eight years with Gould Electronics Systems and Simulation Division five years as an independent consultant, and four years as an Applications Engineer for Phillips Semiconductor He received his BS in Electrical Engineering from Virginia Tech

Ronald G Moore

Evans & Sutherland
600 Komas Drive
Salt Lake City Utah 84158
rmoore@es.com

Ronald G Moore is the manager of the Ground and Helicopter Systems Applications Group at Evans & Sutherland Computer Corporation and the Lead Visual System Project Engineer on the U S Army Close Combat Tactical Trainer (CCTT) program Mr Moore has worked for Evans & Sutherland in the area of real-time visual simulation for eight years He served as a member of the Distributed Interactive Simulation (DIS) Standards Steering Committee and as the Chair of the Simulated Environment Working Group for four years He also served as the interim chair for the Simulated Natural Environment Simulation Interoperability Workshop (SIW) forum of the new Simulation Interoperability Standards Organization

(SISO) Mr Moore is currently on the paper review panel for the Simulated Natural Environment and Sensor forums for the SIW Mr Moore was employed by the Boeing Visual Flight Simulation Laboratory for eight years where he contributed to research in cockpit automation technology and air combat team training using distributed network technology He has been involved in image generation and database development for over fifteen years He received his BSE from Brigham Young University in 1983

Elizabeth L Smith

Paradigm Simulation Inc
14900 Landmark Blvd Suite 400
Dallas TX 75240
eliza@paradigmsim com

Elizabeth Smith is a Director of Engineering for Paradigm Simulation Inc She received an undergraduate degree in Mathematics from Texas Christian University a Masters in Mathematics from California State University Los Angeles and a Masters in Computer Science from the University of Texas at Dallas Before joining Paradigm in 1993 she developed 3D visualization software for Sierra Geophysics and Halliburton Energy Services and was graphics team lead in the Artificial Intelligence Lab and Visualization and Simulation Lab of the (former) Defense Systems and Electronics Group of Texas Instruments Since 1988 she has lectured occasionally in computer graphics at the University of Texas at Dallas and the University of North Texas Her interests include toolkits and interfaces for the development of real time graphics applications as well as techniques for data visualization

Course Syllabus

Time	*Speakers*	*Subject*
8 30	Smith	Introduction, Overview
8 45	Beasley	History and Requirements of Visual Simulation
		Multiprocessing and Graphics Pipelines (Run Time Concepts)
9 00	Beasley	Introduction to Computer Image Generation (CIG) Architectures
9 10	Moore	Evans & Sutherland Architectures for Real-Time Display of Large Areas and Big Textures
9 35	Howie	Ways to Handle Geometry and Textures for Large Geographic Areas in Real-Time
10 00	--- Break ---	
10 15	Clay	Silicon Graphics Architectures for Real-Time Display of Large Areas and Big Textures
10 40	Clay	Run Time Optimization Techniques for Vis Sim Applications
11 00	Beasley	Importance and Requirements of Multispectral Applications
11 10	Gersuk	Real-Time Generation of Multispectral Images
		Visual Database Construction Techniques
11 30	Howie Moore Brockway	Database Designs
12 00	--- Lunch ---	
13 30	Brockway	Large Area Texture and Terrain – "above it all"
14 00	Moore	Integrated Feature Modeling – "stuck in the dirt"
15 00	--- Break ---	
15 15	Howie	Performance Management – "going both ways"
		Displaying the Results
15 40	Gersuk	Special Displays for Vis Sim (Domes, Calligraphics)
16 10	Smith	Creative Uses of Bitplanes
16 30	Panel – ALL	Summary, Predictions, Questions
17 00	-- The End --	

Real-Time Graphics For Visual Simulation:

Advanced Techniques from the Top Down

SIGGRAPH '98 - Course 20

Real-Time Graphics For Visual Simulation

Lecturers

Graham Beasley
Silicon Graphics Inc.

Stephen Gersuk
Paradigm Simulation Inc.

Dan Brockway
Paradigm Simulation Inc.

J. Bruce Howie
Lockheed Martin Information Systems

Sharon Rose Clay
Silicon Graphics Inc.

Ronald G. Moore
Evans and Sutherland

Elizabeth Lee Smith -Course Organizer
Paradigm Simulation Inc.

Real-Time Graphics For Visual Simulation

Descriptions of Terms

- Real-Time
  - *Movement consistent with real world actions*
  - *Updates happen reliably at fixed intervals*
  - *Realistic interaction*
- Visual Simulation - Vis Sim
  - *Abstraction of reality*
  - *Believable to the human eye*

Real-Time Graphics For Visual Simulation

Scope of the Course

- Interactive Computer Graphics for Simulating Visual Phenomena
- Visual Realism, Real World, Reality, Real-Time
- Overview Plus Advanced Techniques
- Software Algorithms on High-End Hardware and the Migration of Vis Sim Techniques from High-End to Low-End
- Large Geographic Areas, Training Systems, Visualization for Engineering Development,...

Real-Time Graphics For Visual Simulation

Outline

- Overview
- Run-Time Issues: Multiprocessing and Graphics Pipelines
- Preparation: Visual Database Construction Techniques
- Final Views: Displaying the Results
- Predictions: Panel

Real-Time Graphics For Visual Simulation

Time Schedule

8:30 Overview

9:00 Multiprocessing and Graphics Pipelines

11:30 Visual Database Construction Techniques

15:40 Displaying the Results

16:30 Predictions: Panel

Real-Time Graphics For Visual Simulation

Time	Speakers	Subject
8:30	Smith	Introduction, Overview
8:45	Beasley	History and Requirements of Visual Simulation
		Multiprocessing and Graphics Pipelines (Run Time Concepts)
9:00	Beasley	*Introduction to Computer Image Generation (CIG) Architecture*
9:10	Moore	Evans & Sutherland Architectures for Real-Time Display of Lar Areas and Big Textures
9:35	Howie	Ways to Handle Geometry and Textures for Large Geographic Areas in Real-Time
10:00	--- Break ---	
10:15	Clay	Silicon Graphics Architectures for Real-Time Display of Large Areas and Big Textures
10:40	Clay	Run Time Optimization Techniques for Vis Sim Applications
11:00	Beasley	Importance and Requirements of Multispectral Applications
11:10	Gersuk	Real-Time Generation of Multispectral Images

Real-Time Graphics For Visual Simulation

Time	Speakers	Subject
		Visual Database Construction Techniques
11:30	Howie Moore Brockway	Database Designs
12:00	--- Lunch ---	
13:30	Brockway	Large Area Texture and Terrain – "above it all"
14:00	Moore	Integrated Feature Modeling – "stuck in the dirt"
15:00	--- Break ---	
15:15	Howie	Performance Management – "going both ways"
		Displaying the Results
15:40	Gersuk	Special Displays for Vis Sim (Domes, Calligraphics...)
16:10	Gersuk Smith	Creative Uses of Bitplanes
16:30	Panel – ALL	Summary, Predictions, Questions
17:00	-- The End --	

Real-Time Graphics For Visual Simulation

Homework - Creative Applications Lab

Real-Time Graphics For Visual Simulation
An Introduction

Elizabeth Lee Smith
Paradigm Simulation, Inc

SIGGRAPH '98 Course 20

Definition

Visual Simulation, under its broadest definition, refers to the techniques and technologies involved in creating images and animations that appear real to the human eye When using the term Visual Simulation (Vis Sim), we usually also mean that images of the simulated visual world are updated in real-time as objects in the world move, and as the observer, whose vision is the focus of the illusion, is made to feel he is moving through the world The emphasis is on visual realism, reality, the real world and interacting with it in real-time

Application Fields

Many application areas for computer graphics seem to fall within the above description A few are architectural walk-throughs, some computer games and entertainment systems, vehicle training systems, and medical imaging It might be easier to list computer graphics fields that are not considered Vis Sim For example, data or scientific visualization programs endeavor to present abstract numerical data with colors, shapes, textures, and movement in order to help the user understand and interpret the data Scientific visualization involves a "realization of the abstract " On the other hand, Vis Sim produces an "abstraction of reality " Trees, terrain, clouds, fire, buildings, people, and vehicles (real world objects) map to polygons, textures, and ultimately, pixels While the computer graphics used in the movie industry achieves visual realism, those procedures are not usually classified as Vis Sim, because they do not provide the intended viewer the capability to interact with the images in real-time

Vis Sim Market

In the commercial world, the term Visual Simulation takes on a more restricted definition Vis Sim, the market, refers to the use of computer graphics in such areas as

- flight and driving simulators,
- training systems of all kinds where the trainee must be aware of and interact with a virtual world,
- testing of engineering prototypes within a simulated world

The context is usually the outdoor world, often covering large

geographic areas This restricted description excludes architectural walk-throughs and surgical simulations from the Vis Sim classification

Vis Sim, the Technology Leader

Numerous computer graphics techniques owe their origin to early real-time Vis Sim projects In particular, flight training was one of the few endeavors worth the cost of simulating real-time interactive realism thirty years ago Today Vis Sim in research and in commerce is still pushing the envelop of what is possible to achieve on a computer, be it a high-end image generator, a workstation, or a personal computer

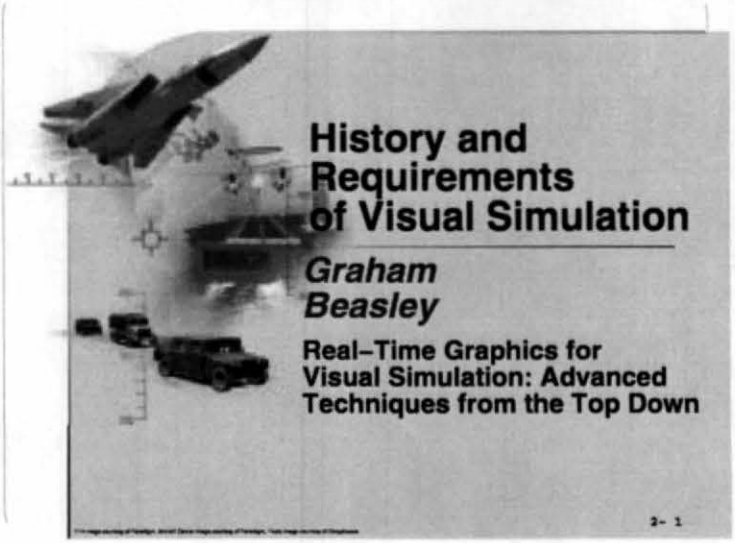
Course Presentation

This course presents a broad overview of the current state of Visual Simulation as well as an in-depth consideration of selected techniques now possible on high-end systems Past trends imply that those techniques will carry over into other fields within computer graphics and migrate down the hardware ladder as less expensive systems become more capable Perhaps this course itself can be an inspiration for furthering the migration.

The course is divided into three sections The section titled Multiprocessing and Graphics Pipelines deals with run-time issues on high-end graphics systems Three different approaches to real-time display of large areas using big textures are presented The section includes a brief survey of multispectral applications, rendering in parts of the spectrum not visible to the human eye The second section, titled Visual Database Construction Techniques, details the preparations needed for and creation methods used in building databases optimized for real-time interactive rendering At the end of the graphics pipeline is the display, and in the final section, Displaying the Results, we discuss framebuffer manipulation algorithms and tricks

We hope that the course will be valuable to participants who are interested in a variety of fields, even though our examples are drawn largely from our work experience

The copies of our presentation slides in these course notes may vary somewhat from those shown during the presentation at the conference in July 1998 Lecturers will give URLs from which any slide updates can be accessed



History and Requirements of Visual Simulation

Graham Beasley

Real-Time Graphics for Visual Simulation: Advanced Techniques from the Top Down

Vis Sim Terminology

Visual Simulation ("Vis Sim")

- Identifies a ***technology***
- Has wide-ranging applications
  - VR, Entertainment, Medical, CFD, Digital Prototyping, etc.

Modeling & Simulation

- Identifies an ***industry***
- Has specific applications
  - Development, Training
  - Government, Civil
  - All in real-time

Trickery and Training

- Major difference between entertainment vs "traditional vis-sim" is realism.
- Entertainment is subjective (eg. DQ ride)
- Traditional vis-sim is usually about training and the training value is more quantitative, (eg. shooting approaches at Denver's new airport in a DC-10).
- Transfer of Training, positive training developing a scan for fighter pilot, negative training looking for the bright targets.

Customer Needs

Critical Requirements and Constraints

- Training
- Engineering Development
- Validation & Procurement
- Mission Planning & Rehearsal
- Safety
- Cost
- Schedule
- Competition

Technical Requirements

- Real–time requirements, eg. latency, update rates, ...
- Display/resolution requirements, eg X HFOV vs Y VFOV, n arc minute resolution (for target identification/recognition), exact number of windows/channels, brigtness/contrast, ...
- Database content driven: scene density, correlation, moving models, depth complexity, articulating parts, special effects, etc.
- No visual anomolies
- Price driven

Market Trends

Market growth tied to price/performance

Apollo Sims	*$50M–$100M*	*1's*
SOF, NASA	*$30M–$50M*	*10's*
Flight, Maritime	*$10M–$30M*	*100's*
Ground Warfare	*$1M–$10M*	*1000's*
Driving	*$100K–$1M*	*10,000's*
Individual	*$10K–$100K*	*100,000's*

The Simulation Market

Examples

Fixed-wing & Rotorcraft Flight

Railroad Metro Subway

Maritime

Space

Air Traffic Control

Ground Warfare

Driving

C4I

Customer Needs

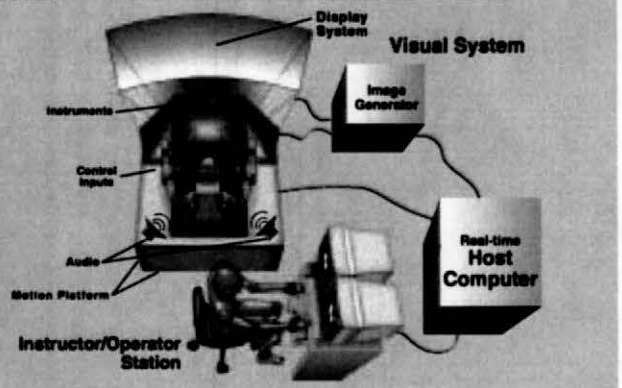
Engineering Simulators

- Human factors design
- Pilot workload studies
- Developmental test and evaluation
- Operational test and evaluation

Training Simulators

- Mission rehearsal
- Full mission
- Part task
- Embedded
- Deployable
- Computer based

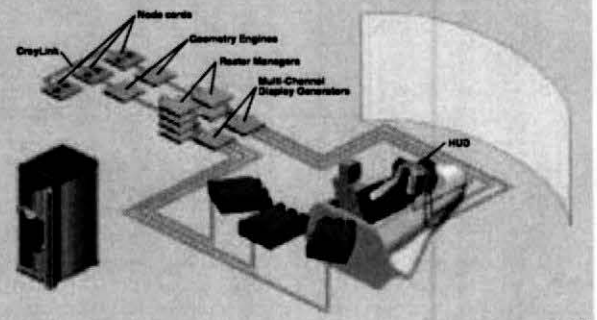
Simulator Components



2-9

Fast Jet Training Simulator

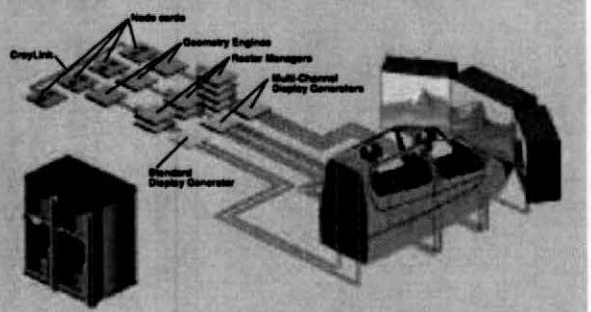
Distributed Mission Training



2-10

Helicopter Simulator

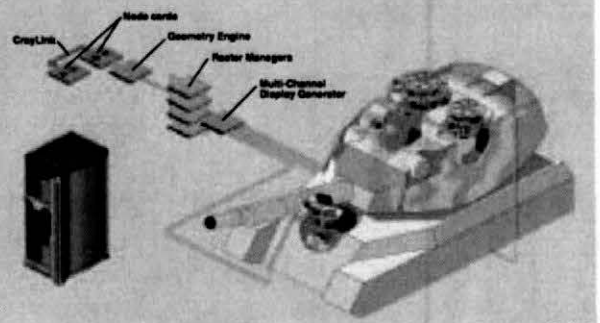
Low-Level Flight



2-11

Tank Simulator

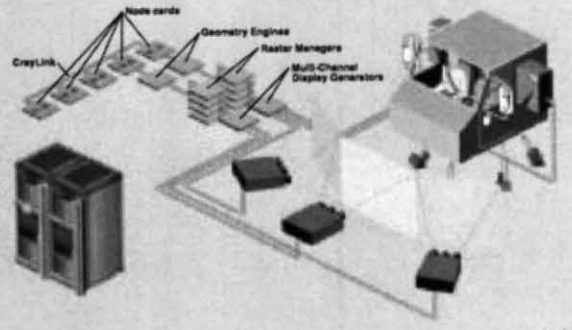
Complex Database Interaction



2-12

Truck Simulator

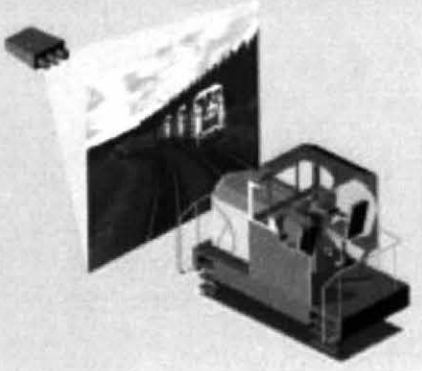
Ground vehicle



2-13

Railway Simulator

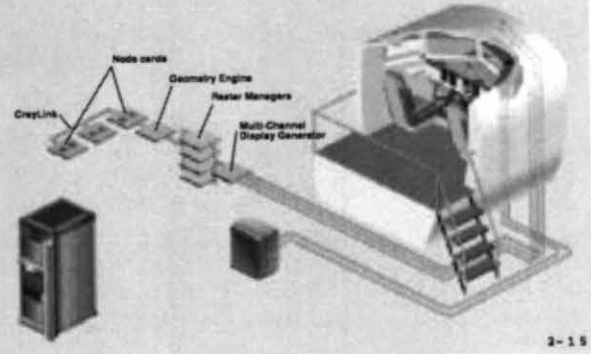
CIG Technology replacing video-disk!



2-14

Civil Airline Simulator

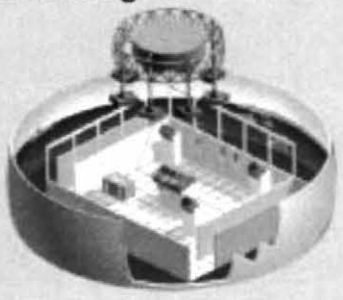
Calligraphics support available



2-15

Ship Bridge Simulator

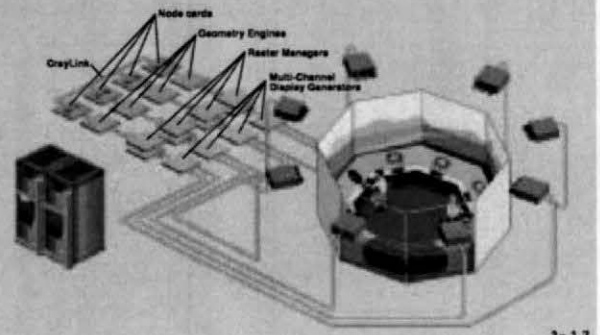
Advanced sea-state modelling increases training realism!



2-16

Air Traffic Control Simulator

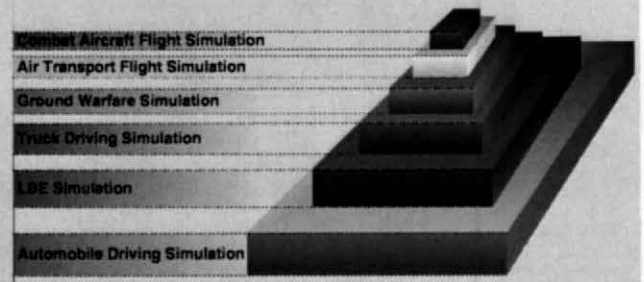
Fixed eyepoint, many moving models



2-17

The Growth of Simulation

Increasing user volume



2-18

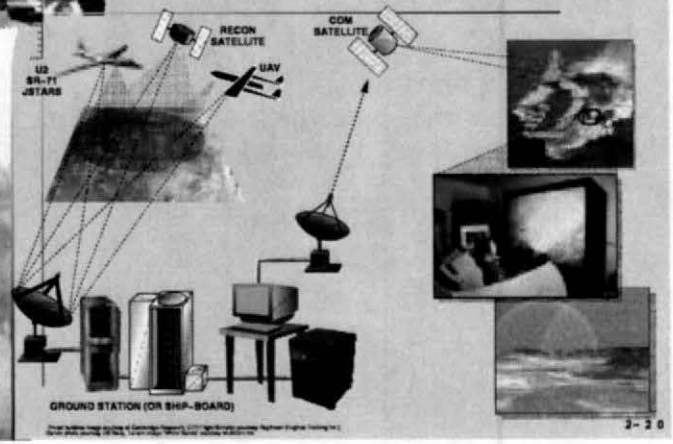
The Future of Simulation

The future looks bright!

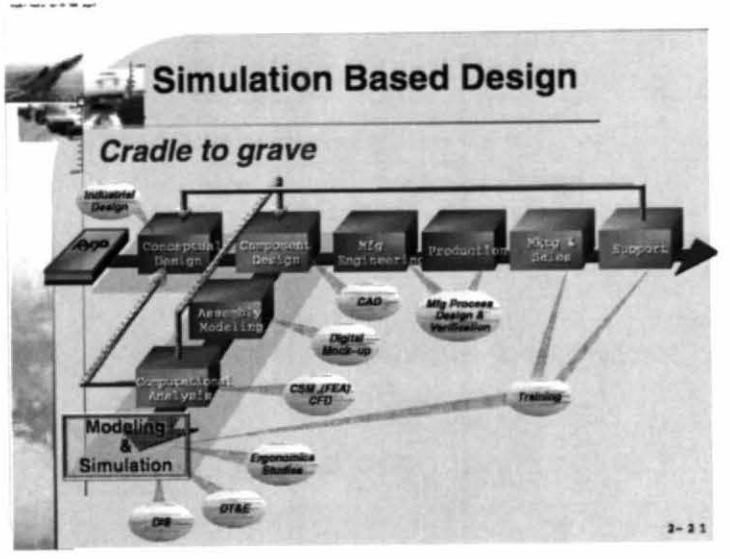
- **The Power of Commercial Systems will Drive Change**
  - Processing speed and *bandwidth!*
  - Data storage and transport
  - Widening markets for simulation technology
- **Simulation Systems Tomorrow**
  - General-purpose computers with multiple applications
  - Open software for modification and reuse
  - Deployed & embedded simulation & training
  - Eye-limiting display resolution--display revolution
  - Simulation and imaging merge

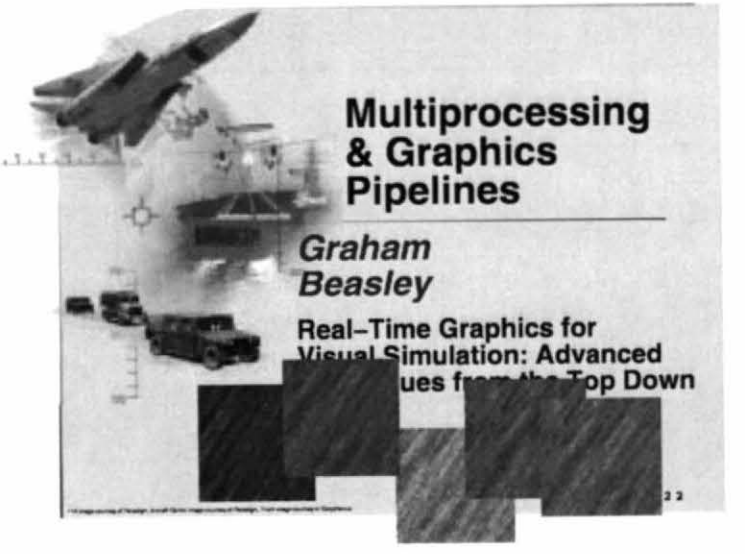
2-19

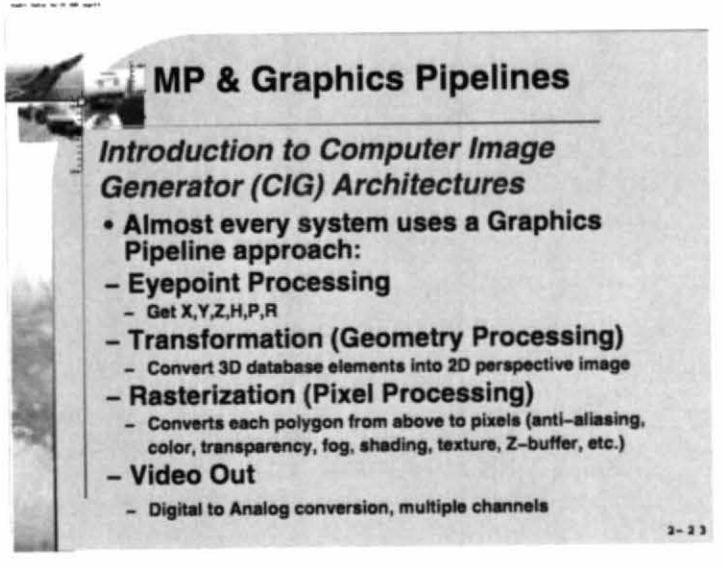
Mission Rehearsal

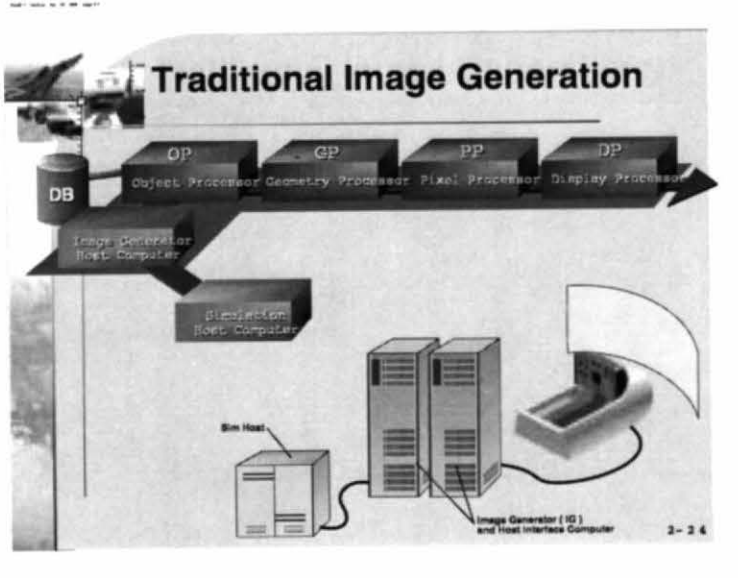


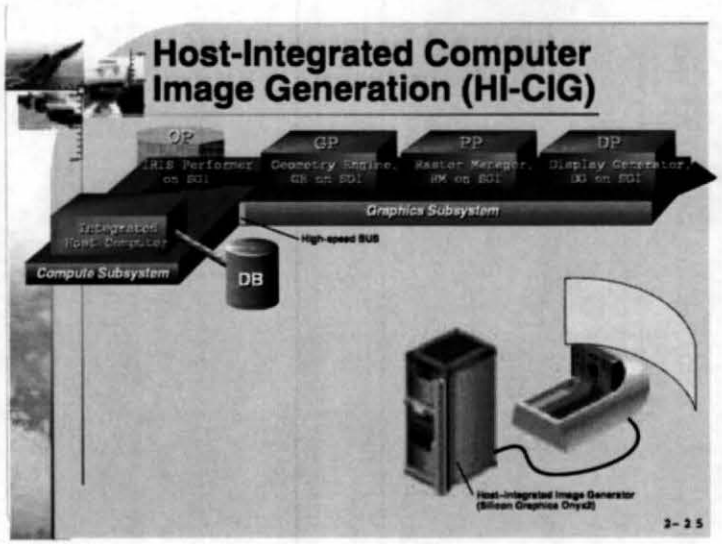
2-20


Simulation Based Design
Cradle to grave
Industrial Design
Concept Design
Component Design
Mfg Engineering
Production
Mktg & Sales
Support
CAD
Mfg Process Design & Verification
Assembly Modeling
Digital Mock-up
CSM (FEA) CFD
Training
Modeling & Simulation
Ergonomics Studies
DB
DT&E
2-21


Multiprocessing & Graphics Pipelines
Graham Beasley
Real-Time Graphics for Visual Simulation: Advanced
Top Down


MP & Graphics Pipelines
Introduction to Computer Image Generator (CIG) Architectures
• Almost every system uses a Graphics Pipeline approach:
– Eyepoint Processing
– Get X,Y,Z,H,P,R
– Transformation (Geometry Processing)
– Convert 3D database elements into 2D perspective image
– Rasterization (Pixel Processing)
– Converts each polygon from above to pixels (anti-aliasing, color, transparency, fog, shading, texture, Z-buffer, etc.)
– Video Out
– Digital to Analog conversion, multiple channels
2-23


Traditional Image Generation
DB
OP
Object Processor
GP
Geometry Processor
PP
Pixel Processor
DP
Display Processor
Image Generator Host Computer
Simulation Host Computer
Sim Host
Image Generator (IG) and Host Interface Computer
2-24


Host-Integrated Computer Image Generation (HI-CIG)
OP
IRIS Performer on SGI
GP
Geometry Engine GE on SGI
PP
Raster Merger RM on SGI
DP
Display Generator DG on SGI
Graphics Subsystem
Integrated Host Computer
High-speed BUS
Compute Subsystem
DB
Host-integrated Image Generator (Silicon Graphics Onyx2)
2-25

A Case Study in Evans & Sutherland Image Generator Architectures

Outline of topics

- Review of the Evans & Sutherland (E&S) Image Generator Architectures
  - *Early systems "Masters of visual trickery"*
  - *ESIG-4000 "Database construction while you watch"*
  - *ESIG-2000/3000/4500 "The last of the tradition"*
  - *Harmony "The top of the Line"*
- Highlight Real-time Visual Techniques

A Case Study in Evans & Sutherland Image Generator Architectures

Tribute to the founders

Ivan Sutherland

David Evans

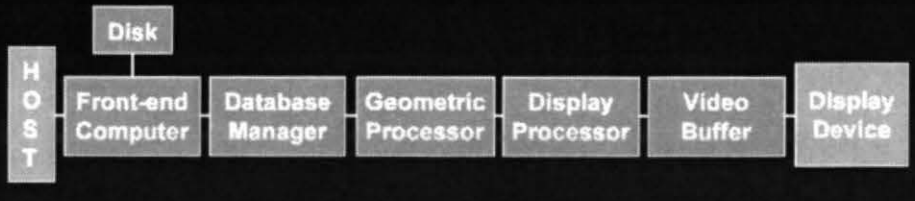
Image Generator Architectures
Early Systems

Masters of Visual Trickery

- Delivered first computer image generator in 1969
- Systems
  - *LDS1*
  - *SP1, SP3T*
  - *NOVOVIEW*
  - *ESIG-LC, ESIG-500, 600*
  - *CT5, CT6, ESIG-1000*

Early Systems
Masters of Visual Trickery

Typical Block Diagram



Early Systems
Masters of Visual Trickery

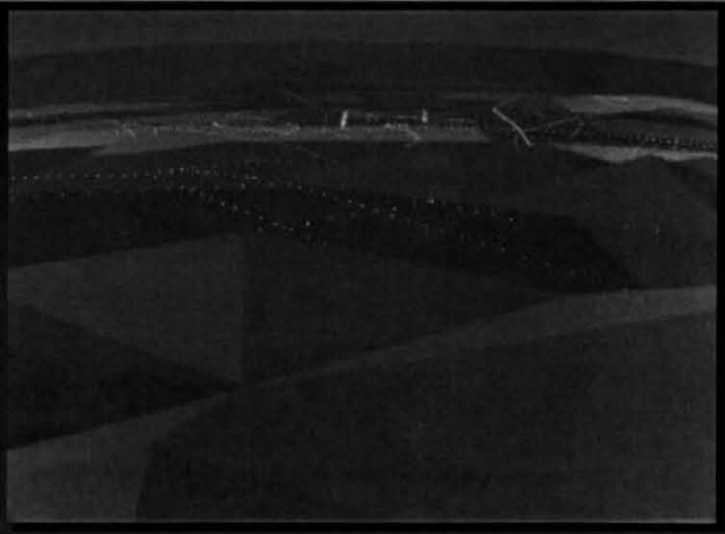
Basic Characteristics

- Fixed Listing and Cellular Priority
- Clamp Texture
- Multiple texture maps per polygon
- Many Unique Texture types
- Pixel Level visibility effects
- Calligraphic lights

Basic Characteristics

- System Fault Monitor
- Many unique cards (25+)
  - *e.g. Clock Maintenance Card*
  - *e.g. Edge File Card*
  - *e.g. Span Generator Card*
  - *e.g. Landing Lobe Card*

Early Systems
SP1 (Early 70s), Night Approach



Early Systems
SP3T (Late 70s), Dusk Scene

Early Systems
CT5 (Early 80s), F-15s

Early Systems
CT5, No Texture

Early Systems
ESIG-500, Low Visibility Approach

Early Systems
ESIG-600, Patchy Clouds

Early Systems
ESIG-1000, F-16 Over SLC

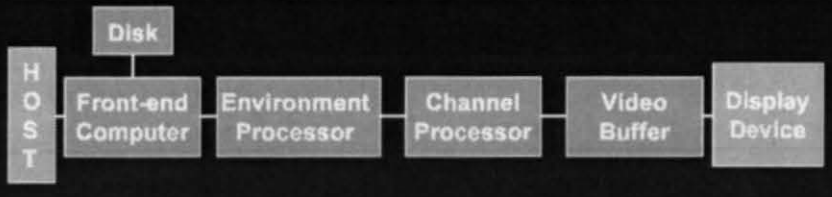
Image Generator Architectures
ESIG-4000

Database Construction while you watch

- Designed for Mission Rehearsal Applications
- Over Ten Billions Arithmetic Operations per second per channel processor

ESIG-4000
Database Construction while you watch

Block Diagram



ESIG-4000
Database Construction while you watch

Basic Characteristics

- Run-time generation of terrain skin from elevation grids
- Run-time placement of 3D models from vector feature data
- Real-time conforming of 3D models to the terrain skin
- Fully Range Buffered

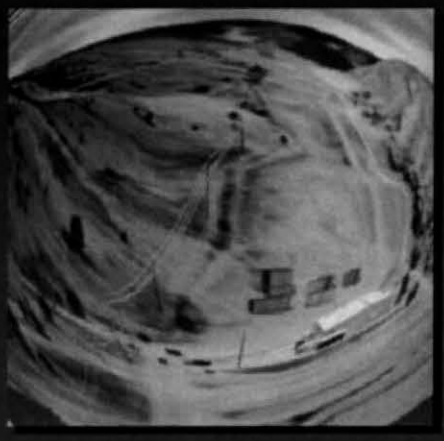
Basic Characteristics

- Built-in distortion correction for complex display systems
- Betweening (morphing) for Level of Detail management
- Themel Sets
- Global Texture

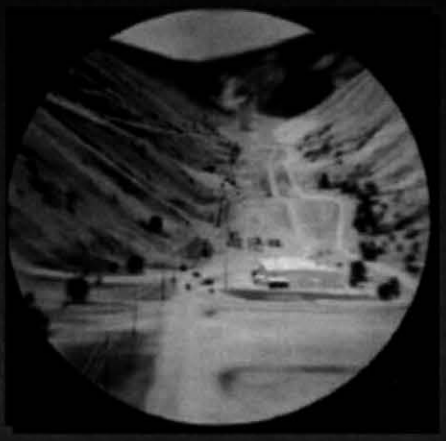
ESIG-4000
Mission Rehearsal

ESIG-4000
Point Defiance Bridge

ESIG-4000
Distortion Correction



Pre-Distorted Image



Dome View

Image Generator Architectures
ESIG-2000/3000/4500

The last of the tradition

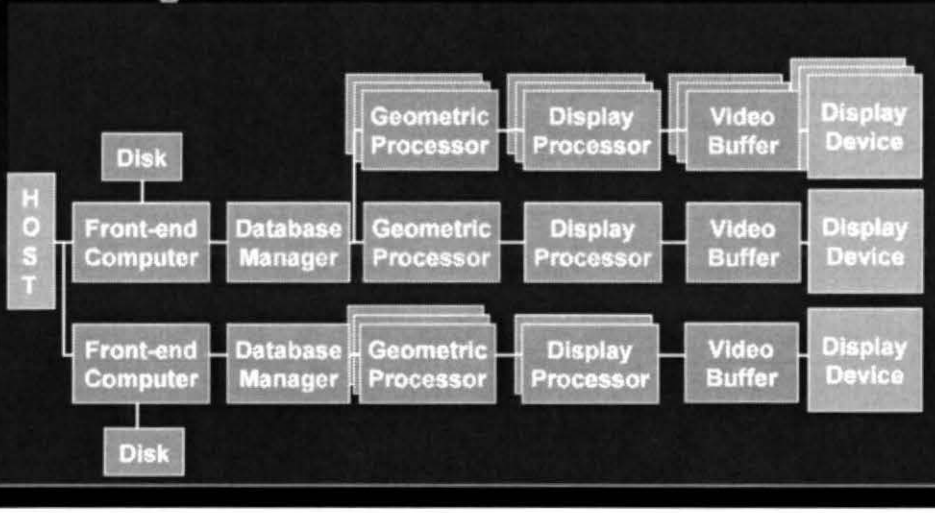
Liberty

ESIG-4520

ESIG-4530

ESIG-2000/3000/4500
The last of the Tradition

Block Diagram



ESIG-2000/3000/4500
The last of the Tradition

Basic Characteristics

- Fix List, Binary Separating Plane processing and Range buffered
- Magnificent Screen Coverage Optimization Techniques
- Basis Sets
- Real-time conforming of 3D models to the terrain skin
- Texture Paging

Basic Characteristics

- Built-in distortion correction
- Fade Level of Detail
- Level of Detail Management
- Stamps
- On-chip Sensor Post Processing
- Pixel Panning
- Parallax

ESIG-2000
Dusk Combat Scene

ESIG-2000
AH-1 Autumn Scene

Real-time Visual Technique
Stamps

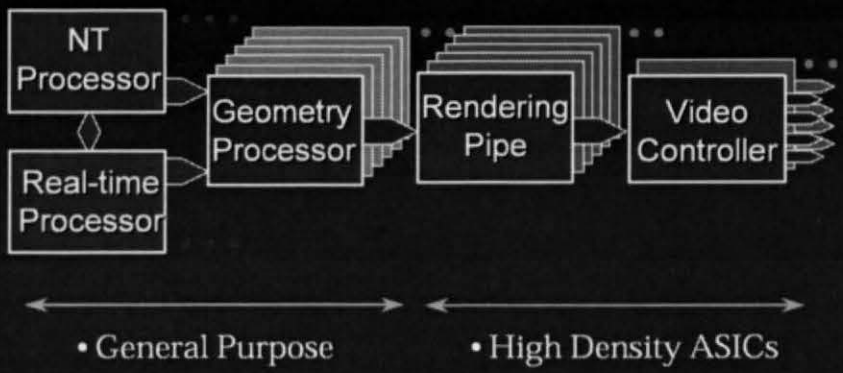
- Maintains face orientation to the eye
- Rotates about vertical axis
- Optimization of stamp groups in distance
- Multiple 'trees" per stamp

Image Generator Architectures
Harmony

The top of the Line

Harmony
The top of the Line

Block Diagram



Harmony
The top of the Line

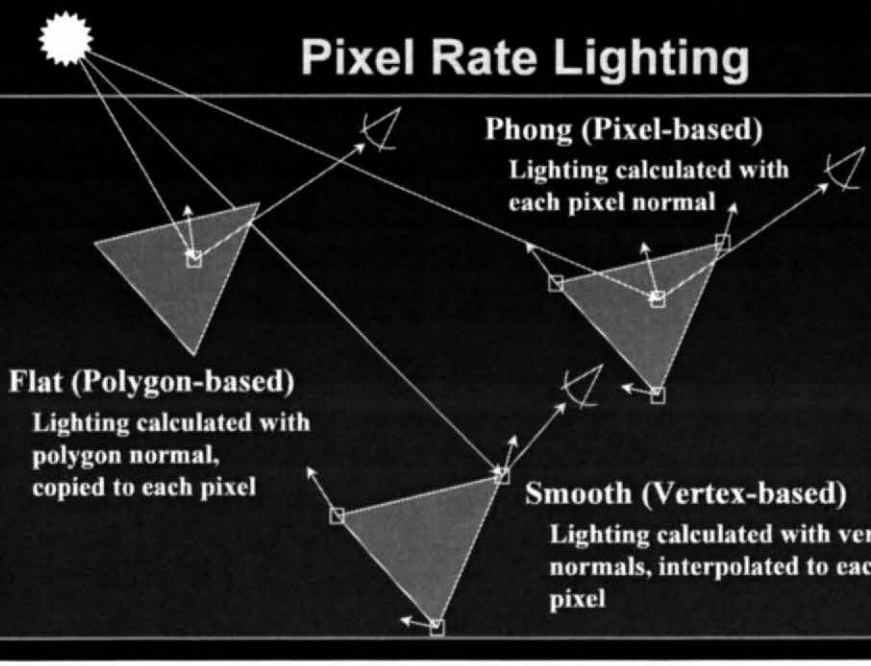
Basic Characteristics

- Pixel rate Lighting model
- Bump Maps
- Super Texture (anisotropic filtering)
- Referenced texture
- Unordered Transparency
- OpenGL

Basic Characteristics

- Coupling of Modeling tools and Run-time system
- Parametric modeling
- On-board sensor effects
- Morphing of terrain and features
- Physic based smoke
- Textured layered fog

Pixel Rate Lighting



Gouraud Shading

Phong Shading

Phong Shading with Bump Map texture

Standard MIP texture

MIP Texture with SuperTexture

Textured Layered Fog

Real-time Visual Technique
Global Texture

Global Texture

- Geospecific
- 22 MIP levels
- Pico (Micro) texture

Themel Texture

- Feature data to select theme elements (texture with 3D models)

Real-time Visual Technique
Themel Texture

1325 polygons with global texture

1365 polygons without global texture

Visual Simulation on SGI Graphics Pipelines

SIGGRAPH 98

Sharon Rose Clay, Scene Graphs
Silicon Graphics

SGI Tools for Visual Simulation

SIGGRAPH 98

Hardware Product Line

- Complete Visual System
  - Configuration Range $<10K–$?M
  - Multiple CPUs, Graphics Pipelines
  - Scalable high–bandwidth I/O
- Integration with 3rd–Party Hardware
  - Calligraphic light points
  - Hi–res displays

SGI Tools for Visual Simulation

SIGGRAPH 98

Software APIs & Toolkits

- Low Level: OpenGL®
- Database Description (Scene Graph)
  - IRIS Performer™, Optimizer™, ImageVision™, Inventor™
- ISVs provide additional pieces for full solution

SGI Tools for Visual Simulation

SIGGRAPH 98

Techniques for Real–Time 3D Graphics

- Symmetric shared memory multiprocessing
- High bandwidth: infinite and dynamic databases
- General hardware: utilization/trade-offs
- Hardware differentiation under standard APIs
  - "Fahrenheit" (project code name)
- Real–Time Diagnostics

Multiprocessing Issues

Pipelining vs. Parallelism

- Task Assignment (Coarse-grained)
- Throughput vs. Latency & Synchronization
- Utilization vs. Bottlenecks, Load-balancing
- Mutual Exclusion and Frame Coherency
- Debugging: Real-Time Diagnostics

Multiprocessing

Task Assignment: Pipelined

- Rendering and Scene management easy to pipeline (APP_CULL_DRAW)
  - APP: Viewpoint , Object Positions
  - CULL: Frustum test -> Occlusion test -> LOD -> Rendering order
  - DRAW: Background, Draw Scene, Scan-out

Multiprocessing

Task Assignment: Parallelized

- Asynchronous or non Real-Time Computation
- Pipelines can be parallelized
  - Speculative computation
  - Multiple Viewpoints, Stereo
  - Multiple Scene Graphs
  - Multiple Render Areas

Multiprocessing

Latency & Synchronization

- Longer Pipeline -> longer latency
- More parallelism -> more synchronization

Multiprocessing

Managing Shared Memory for Scalability

- Mutual Exclusion: copies vs. synchronization
- Frame Coherency: copies are required
- More parallelism –> more synchronization
- Address space: explicit shared arenas moving to implicit shared (POSIX)
- Think CPUs: 4 –> 1000s

Multiprocessing

Synchronization

- Synchronize frames to video refresh
- Synchronization of hardware requires hardware
  - Genlock for tying video refresh of multiple video channels
  - Gang swap for multiple pipelines to double–buffered framebuffers

Infinite Database Management

- Run–time per–frame center of Interest
- Automatic paging – hierarchical (parametrized/customizable/replaceable)
- Hi–resolution insets
- Continuous Level of Detail (LOD) with smooth transitions
- Integrated with frame and load management
- Texture and geometry decoupled

Database Management: ASD Terrain

Active Surface Definition(ASD) Geometry

- Run–time hierarchical evaluation structure
- Morph between pre–computed LODs for continuous and smooth LOD transitions
- Asynchronous evaluation
- LOD evaluation is user–defined (constrained to avoid cracks)
- Direct support for frame/load management

Database Management: ASD Terrain

Building an ASD

- Start from any data input format
- Triangulations can/should be irregular and based on real data points
- Build top-down (low-res to highest)
- Mark key points that should not morph
- Use matching evaluation function

Commercial ASD tools available

Database Management: ASD Terrain

In-Memory Data Structure Hi-lights

- Data points are stored uniquely
- Morph control (weight, morph vector) is per-vertex
- Triangle-strip information pre-computed
- Triangles know LOD level
- Triangle bounding information (for paging)
- Triangles can refer to local graphics state

Database Management: ASD Terrain

ASD Run-Time Evaluation

- Asynchronous evaluation based on frustum (bloated), and evaluation function
- LODs of tiles paged in as needed
- Multiple evaluations (eg. viewpoints) tied or independent
- Top-down evaluation for load control
- Weaving evaluation for frame coherence

Database Management: ASD Terrain

ASD Run-Time Queries and Alignment

- Object points can be automatically aligned
- Objects can be automatically re-tessellated to match terrain
- Query points get back alignment and timing information for extrapolation of moving objects

Database Management: Cliptexture

Cliptexture: Beyond MIPmapping

- Virtualized texture memory
- Efficient use and paging of texture memory
- Removes constraints on geometry
- Continuous and smooth LOD is implicit
- Well suited for load-management
- Automatic alignment of cultural features and moving objects

Database Management: Cliptexture

Cliptexture: Setting it up

- Create tiled LODs of image data
- Specify size and depth of clipped stack
- Link geometry, viewer, and texture memory resources (and potential slave resources)
- Create centering node (default or custom)

Database Management: Cliptexture

Cliptexture: Run-Time

- Center of interest is determined and mapped to texel space
- Areas around COI are scheduled for asynchronous paging (low res first, sorted around COI)
- Texels paged limited by specified time
- Max valid LOD specified

Database Management: Cultural Features

Texture for Cultural Features

- Replace Geometry
- Sharpen cookie-cut textures (trees, fences)
- Correct viewing aspect ratio with MIPmap LOD bias
- Multipass: landing lights, shadows, range-correct fog

Database Management: Cultural Features

Texture Management for Cultural Features

- Palettize
- Keep small number of texture sizes and shapes
- Subload in fixed sizes where possible
- Benchmark paging rates

Database Management: Cultural Features

Flux/Morph for Dynamic Features

- Efficient LOD
- Character animation
- Geometric alignment
- Sync-groups for automatic frame management for asynchronous computation

Frame and Load Management

Closed loop system

- Input: Timing information and predictions and current per-channel stress level
- Select: Geometric and Raster LOD
  - Geometry: pop, fade, morph
  - Raster: Dynamic Video Resize
  - Infinite resolution LOD: calligraphic points
- Measure rendering time

Diagnostics

Integrated Diagnostics

- Graphics pipeline: operational counts, depth complexity, and timing of rendering
- Scene graph: traversal statistics and timing
- System: memory usage, processor utilization

Summary

Real-Time Graphics for Visual Simulation

- Full Visualization System Approach
- Configurable
- SW standards with HW differentiation

Demos !!!

Developing Visual Simulation Applications on Graphics Pipelines

Sharon Rose Clay
Silicon Graphics Computer Systems

Real-Time Graphics for Visual Simulation[1]
SIGGRAPH '98 Course

1 Introduction

VisSim — not just for the ultra high end anymore

Visual Simulation(VisSim) applications have been using real time graphics for the training of pilots of many types of expensive and dangerous craft since the 1960's Originally these systems were very specialized and the technology was esoteric Over time, these techniques have become refined and advances have been made in commercially available hardware and software so that off the-shelf solutions can be used for the pilot training and mission rehearsal tasks that once requiredvery custom solutions The wider availability of cheaper training solutions has broadened the visual simulation market to serve not just training of pilots of jet fighters, but the operators of helicopters ships cranes, trucks trains surgical equipment and just about anything where a mistake is as expensive as available suitable training technology Thus it is no surprise that visual simulation applications are very sensitive to image quality performance, and system cost Commercial graphics hardware platforms provide full product lines with a full range of price points and performance options High end graphics workstations provide many traditional Image Generator features such as real time global texture mapping and full scene antialiasing They can also support many channels

1 Some of the techniques and methods disclosed here are covered by patents held by Silicon Graphics including U S Patent Nos 5 051 737 5 369 739 5 438 654 5 394 170 5 528 737 5,528 738 5 581 680 5 471 572 and patent applications pending We encourage the use of these techniques on Silicon Graphics systems but this functionality is not available for re implementation or distribution on other platforms without the explicit permission of Silicon Graphics

and players per workstation to share resources and offset the cost of the high-end features They also provide sophisticated development environments At the low end, PCs are further broadening the visual simulation market while steadily incorporating more of the proven techniques from the high end The once esoteric innovations of pioneers have become proven methods and adapted and refined for hardware that can be mass produced The growing acceptance and popularity of standard Application Programming Interfaces (APIs) for 3D graphics, such as OpenGL® is making it possible to develop applications that are portable between vastly different architectures Performance however, is typically not portable between architectures so an application may still require significant platform-significant The use of higher-level APIs can greatly reduce the complexity of the application and allow the underlying API to deal with the details and special features of the current hardware For the Silicon Graphics product line this has been solved with a scene graph level software application layer IRIS Performer™ which is specifically targeted at real time 3D graphics applications and is used for examples for many of the techniques discussed in this paper [Rohlf94] Writing/tuning rendering software is discussed in Section 5

High level APIs allow SW portability and HW differentiation

Thus, it is possible to make some useful generalizations about techniques for visual simulation and graphics hardware characteristics that aid in developing and tuning real-time visual simulation applications One key trend is that the demand for better image quality and thus greater graphics resources has been insatiable (to the delightof graphics engineers) In particular, multiprocessing of both the application and of the graphics processing has been a key element of the success of commercial hardware Graphics processing has naturally lent itself to the pipelining of the graphics operations across multiple processors Antialiasing texture mapping, morphing, and level of detail have allowed for high quality images with rich scene content Texture mapping has been the basis of many techniques for achieving heightened realism

This section of the SIGGRAPH '98 course "*Real Time Graphics for Visual Simulation*" presents a strategy for performance for developing and tuning real-time graphics applications using these techniques on commercial graphics pipelines Much of this paper focuses on graphics workstations as they have had more of a history with the many issues that are now making their way down into the low end PC solutions

The following topics are covered

- Typical visual simulation requirements
- Multi processing issues for graphics subsytems
- Performance issues in graphics systems

- Tuning real time graphics applications
  - Graphics workstation pipelines and performance trade offs
  - Strategies for diagnosing pipeline bottlenecks
- Techniques for Visual Simulation Applications
  - Managing frame rate and latency
  - Level of Detail
  - Effective and efficient uses of texture
  - Database structure for traversal
- Run time diagnostics
- Tools for debugging graphics performance

Developing a designed for performance visual simulation application requires under standing the potential performance problems identifying which factors are limiting per formance and then making the trade offs to achieve maximum frame rate with the highest quality scene content

2 *Visual Simulation Requirements*

One of the most important parameters in the effectiveness of a simulated environment is *frame rate* — the rate at which new images are presented The faster new frames can be displayed, the smoother and more compelling the animation will be Constraints on the frame-rate can determine how much time there is to produce a scene [1]

2 1 Frame Rate and Latency Requirements

Frame rate is key component of image quality

Immersion = f(frame rate latency)

Visual simulation applications typically require at the low end a frame rate of at least 20 frames per second (fps) and more commonly 30fps High-end simulation applications such as flight trainers will accept nothing less than 60fps If for example, we allow two milliseconds (msecs) of initial overhead to start frame processing, one msec for screen clear or back-ground and two msecs for a window of safety a 60fps application has, optimistically, about 11 msecs to process a frame and a 30fps application has 28 msecs

Another important requirement for Visual Simulation applications is mini-mizing *total system latency* — the time from when a user event occurs, such as change of view to an input device, to the time when the last pixel

1 There are also additional requirements associated with frame rate such as low variability of frame rates and the handling of overload conditions when frame rates are missed These issues are dis cussed later in Section 11

for the corresponding frame is displayed by the video output on the screen The display of a complete video image is called a *frame* A frame is an integral multiple of the video refresh rate and each video retrace period is called a *field* Video is often running at a 60Hz field rate and application frame rates may be running at 60Hz or 30Hz, etc Interlaced applications run at best at half the field rate with even and odd fields scanning out even and odd lines of the final image, respectively When using interlaced video where alternate video fields display the even and odd lines respectively of a complete frame, the definition is extended to specify the last pixel of the first field of the new frame In visual simulation systems this is often also referred to as *transport delay* High latency is intolerable by human subjects and result in simulator sickness[Kenn90][Kenn92] Latencies of higher than 120msecs are usually considered unacceptable in an immersive application[Cardul90] and 60-85msecs (3 5 50 or 60Hz video fields) is usually the required range specified Given that one field will be used for video scan-out, this leaves 2-4 fields (assuming 60Hz fields) for input device latency, application processing and rendering Latencies tend to be most noticeable and most bothersome on the very actions that are likely to cause higher latency quickly panning the view from side to side causing drastic and sudden change in the visible scene Methods for addressing sys tem latency are discussed later Reducing Latency ' on page 39

Keep latency under 85 msecs

2 2 Rendering Performance Requirements

The gating element of system performance for a graphics application should be the rendering performance of the system Estimating expected system performance is one of the most basic and most important elements of designing and later tuning a real-time graphics application The basic graphics elements that contribute to the time to render a frame are

- screen clear (color and z buffer clear or reset)
- amount of data transferred to the graphics subsystem
- selected attributes for geometry such as lighting texturing atmospheric effects and number of different attribute sets
- viewing transformations of geometry
- the number of pixels produced for the frame (resolution multiplied by *depth complexity*)
- video refresh to output final image from framebuffer memory

An estimation of expected performance should take into account all of these frame components, plus possible overhead due to interactions between components An estimation of the time in milliseconds required to render a frame will then translate into an expected frame rate

Screen clear time is like a fixed tax on the time to render a scene and for rapid frame rates may be a measurable percentage of the frame interval Because of this most architectures have some sort of screen clear optimization For example the Silicon Graphics RealityEngine™ has a special screen clear that is well under one millisecond for a full high-resolution framebuffer (1280x1024) and the InfiniteReality™ made this capability more general to normal screen clears and fast fill rates for background polygons (drawn with depth function to ALWAYS [write] to force values into the framebuffer) Video-refresh also adds to the total frame time and is discussed in Section 4

The size and contents of full databases vary tremendously among different applications However for context we can make a reasonable hypothetical guess at scene characteristics given the high frame rates required for real-time graphical applications and current capabilities of graphics workstations

Geometry

Match requirements against expected performance

The number of polygons possible in a 60fps or 30fps scene in commercial graphics subsystems is affected by many factors and choices in database design but needless to say it can be quite different than the peak polygon transform rate of a machine This is a large break from traditional high end image generators where there was enough customized hardware present to ensure a guaranteed rate for any type of polygon Current graphics workstations have more generalized hardware that is always fully utilized and offers fast and slower modes for different types of rendering This offers the power to make performance/image quality trade offs but makes the task of performance prediction much more complicated Typical current rates fall somewhere in the large range of 3000 and 10000 triangles at 60pfs Typical attributes specified for triangles include some combination of normals colors, texture coordinates, and associated textures For visual simulation applications, the amount of dynamic objects and geometry changing on a per-frame basis is typically relatively high For handling general dynamic coordinate systems of moving objects, matrix transforms are most convenient Such objects usually also have relatively high detail Simple objects may have 50-100 polygons Animated characters can have tens of thousands of polygons at their highest level of detail particularly for entertainment applications where the characters are a major focus The number of active moving objects has become very high with the advent of Distributed Interactive Simulation (DIS) battlefield training where on the order of 10K active moving entities may need to be tracked and several hundred may be in the current frame

These numbers imply that we can easily imagine having a megabyte of just geometric graphics data per frame Demands for texture data can be equally high with on the order of several hundred KBytes for fast flight and drive applications to a MByte of texture data needed per frame for a fast moving /ooming from-the-moon-to-the-street type of application Disk paging and host-graphics bandwidth are essential to these applications In long range flight applications and applications using geospecific data areas are very large and new data may even be received in real-time during the running of the application Thus, it is often unreasonable or even impossible to store the entire database in memory and the data must be paged in as the viewer moves

Raster

Consider depth complexity in a ddition to resolution to compute fill requiremetns

Visual simulation applications tend to be very demanding of the pixel pro cessing part of the graphics pipeline for operations like texture mapping *Depth complexity* is the number of times, on average, that a given pixel is written A depth complexity of one means that every pixel on the screen is touched one time This is a resolution-independent way of measuring the fill requirements of an application High-altitude applications tend to have depth-complexity between two and three, ground based applications tend to have depth complexity between three and five Depth complexity can be reduced through aggressive database optimizations, discussed in Section 9 Resolutions for visual simulation applications also vary widely For low-end visuals, NTSC(640x480) interlaced resolution used to be common but has been replaced with VGA (non-interlaced) and 800x600 is becoming more the low-end standard At the high end, the display system is usually considered the limiting factor With the arrival of laser projectors what is called eye-limiting resolutions of up to 5Kx4K are being demanded which requires both high fill rates and high video bandwidth A 60fps application at that resolution (non-interlaced) with depth-complexity five will require a fill rate of 6 GPixels/sec

The published specs of the machine can be used to make a rough estimate of predicted frame rate for the expected scene content However, this prediction will probably be very optimistic Performance prediction is covered in detail in Section 5 An understanding of the graphics architecture enables more realistic calculations

The type of system resources available and their organization has a tremendous effect on the application architecture A common thread is that multi processing of some form has been a key component of the high-performance graphics platforms and is working its way down to the low-end platforms Architecture issues for graphics subsystems are discussed in detail in Section 3 and Section 4

3 Multi-Processing for High-Performance Graphics

A significant part of designing and tuning an application is determining the best way to utilize the system processing resources and determining if additional system resources will improve performance Every tuning strategy requires an understanding of how the different components in a system interact to affect performance There are many elements to system performance beyond the guiding frames-per-second

Throughput — maximizing frame rates is equivalent to maximizing throughput producing the most output possible per unit of time

Bandwidth — The major datapaths through the system must have sufficient bandwidth or entire parts of a system may be underutilized The connections of greatest concern would be 1) that between the host computer and the graphics subsystem 2) the paths of access to database memory and 3) disk access for the application and the graphics subsystem It is particularly important that bandwidth specs not assume tiny datasets that will not scale to a real application

Processor utilization — Will the system get good utilization of the all available hardware or will some processors be sitting idle while others are overloading (will you get what you paid for) Good processor utilization is essential for a system to realize its potential throughput Achieving this in a dynamic environment requires load balancing mechanisms

Scalability — If performance is a problem will the system support the addition of extra processors to improve throughput and will improved performance scale with the addition of new processors? Additionally as new processors are added will load balancing enable a real application to see the improved performance or will it only show up in benchmarks?

Latency — What is the maximum interval of time from when a user initiated an input and the moment the final pixel of the corresponding new frame is presented? Low latency is critical to interactive real time entertainment applications

Synchronization overhead — how much overhead is incurred when tasks communicate information This is particularly an issue for the very dynamic database of an interactive real time visual simulation application both the main application and the graphics subsystem need efficient access to the current state of the database

These measures of performance can be applied to both the system as a whole and to individual subsystems

3 1 Methods of Multiprocessing

Because graphics applications have many very different tasks that must be executed every frame they are well suited to division among multiple tasks, and multiple processors if available Multiprocessing can also be used to achieve better utilization and throughput of a single processor

Pipelining vs. Parallelism

The partitioning and ordering of the separate tasks has direct consequences on the performance of the system A task may be executed in a pipelined or in a concurrent fashion *Pipelining* uses an assembly-line model where a task is decomposed into stages of operations that can be performed sequen tially Each stage is a separate processor working on a separate part of a frame and passing it to the next stage in the line *Concurrent* processing has multiple tasks simultaneously working on different parts of the same input producing a single result

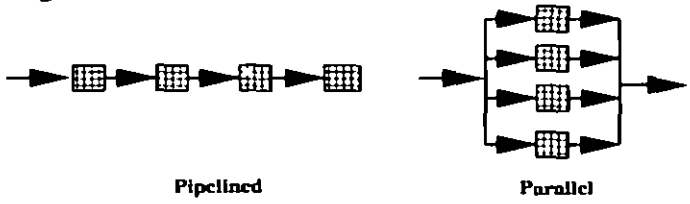


FIGURE 1 Pipelined vs Parallel Processors

Parallelism has lower latency but needs more synchronization

Pipelining has higher latency but easier to add and requires less synchronization

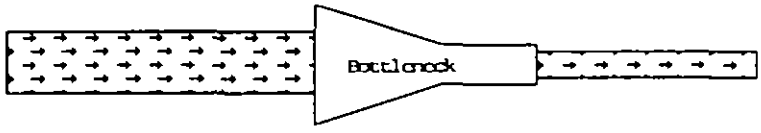
Both the host and graphics subsystems may employ both pipelining and parallelism as a way of using multiple processors to achieve higher performance The general theoretical multiprocessing issues apply to both graphics applications and graphics subsystems Additionally, there are complexities that arise with the use of special purpose processors, and from the great demands of graphics applications

Many graphics tasks are easily decomposed into pipelined architectures Typically there is a main graphics pipeline, with parallelism within stages of the pipeline Individual pipeline stages may themselves be sub-pipelines or have parallel concurrent processors Additionally, there may be multiple parallel graphics pipelines working concurrently In the case of the Silicon Graphics platforms mutliple graphics pipelines could be connected directly to the same host to allow themto share host memory and processors and to allow one application to drive the entire system

Pipelined systems need minimal synchronization because each task is working on its own data for a different frame or part of a frame, in an ordered fashion so synchronization is implicit Longer pipelines have increased throughput — producing new results in quick succession because the task has been broken up into many trivial stages that all execute quickly However, each stage in a pipeline will add latency to the system because the total amount of time through the pipeline is the number of stages multiplied by the step time, which is the speed of the slowest stage While every step produces a new output, the total amount of time to produce a single complete output may get longer The addition of a new pipe

line stage will presumably decrease the step time but probably not enough to avoid overall increased latency

Pipelined systems will always run at the speed of the slowest stage, and no faster The limiting stage in a pipelined system is appropriately called a *bottleneck*



Pipeline tuning amounts to determining which stage in the pipeline is the bottleneck and reducing the workload of that stage This can be quite difficult in a graphics application because through the course of rendering a frame the bottleneck changes dynamically Furthermore, one cannot simply take a snapshot of the system to see where the overriding bottleneck is Finally, improving the performance of the bottleneck stage can actually reduce total throughput if the another bottleneck results elsewhere Bottleneck tuning methods are discussed in Section 5

Tune the slowest stage of the pipeline

Concurrent architectures do not suffer from the throughput vs latency trade-off because each of the tasks will directly produce part of the output However synchronization and load-balancing are major issues If processors are assigned to separate tasks that can be run in parallel then there is the chance that some tasks will take very little time to complete and those processors will be idle If a single task is distributed over several processors then there is the overhead of starting them off and recombining the output results However, the latter has a better chance of producing an easily-scalable system because repetitive tasks such as transforming vertices of polygons can be distributed among multiple concurrent processors Concurrent parallel architectures are also easier to tune because it is quite apparent who is finishing last

SIMD vs. MIMD

The processor organization in the system also needs to be considered There are two types of processor execution organization SIMD or MIMD SIMD (single instruction multiple data) processors operate in lock-step where all processors in the block are executing the same code These processors are ideal for the concurrent distributed-task model and require less overhead at the start and end of the task because of the inherent constraints they place on the task distribution SIMD processors are common in graphics subsystems However, MIMD (multiple instruction multiple data) do better on complex tasks that have many decision points because they can

each branch independently As with pipelined architectures the slowest processor will limit the rate of final output

In actual implementation, graphics architectures are a creative mix of pipe lining and concurrency There may be parallel pipelines with the major pipeline stages implemented as blocks of parallel processors

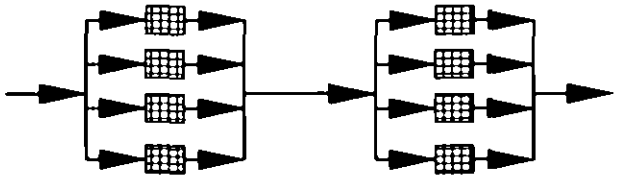


FIGURE 2. Parallel Pipeline

Individual processors may then employ significant sub-pipelining within the individual chips Systems may be made scalable by allowing the ability to add parallel blocks

4 Performance Issues in Graphics Pipelines

This section briefly reviews the basic rendering processes in the context of presenting the basic computational requirements Additionally, ways in which the rendering task can be partitioned for implementation in hardware and corresponding performance trade-offs are also discussed Tuning an application to a graphics pipeline is discussed in detail in Section 5

The task of rendering three-dimensional graphics primitives is very demanding in terms of memory accesses, integer, and floating-point calculations There are impressive software rendering packages that handle three dimensional texture-mapped geometry and can generate on the order of 1MPixels/sec on current CPUs However, the task of rendering graphics primitives is very naturally suited to distribution among separate, specialized pipelined processors Many of the computations that must be performed are also very repetitive, and so can take advantage of parallelism in a pipeline This use of special-purpose processors to implement the rendering process is based on some basic assumptions about the requirements of a typical target application The result can be orders of magnitude increases in rendering performance

4 1 The Rendering Pipeline

The rendering process naturally lends itself to a simple pipeline abstraction The *rendering pipeline* can generally be thought of as having three main stages

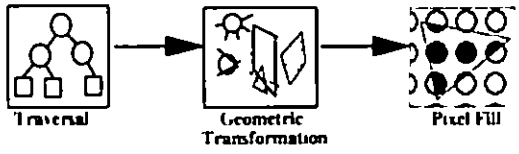


FIGURE 3 The Rendering Pipeline

Each of these stages may be implemented as a separate subsystem These different stages are all working on different sequential pieces of rendering primitives for the current frame A more detailed picture of the rendering pipeline is shown in Figure 4 An understanding of the computations that occur at each stage in the rendering process is important for understanding a given implementation and the performance trade-offs made in that implementation The following is an overview of the basic rendering pipeline, the computational requirements of each stage and the performance issues that arise in each stage[1][Foley90 Akeley93,Harrell93 Akeley89]

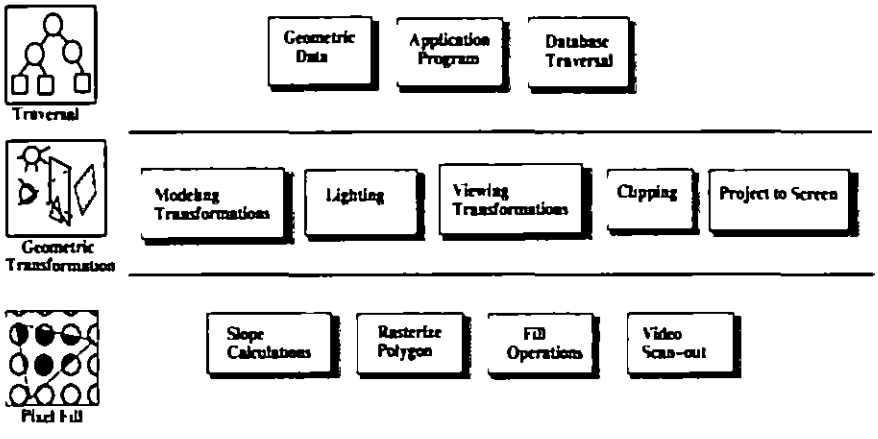


FIGURE 4 The Detailed Stages of the Rendering Pipeline

1 See [Foley90] Chapter 18 for a detailed discussion of this topic

4 2 The CPU Subsystem (Host)

At the head of the graphics pipeline is the main real-time application running on the host If the host is the limiting stage of the pipeline the rest of the graphics pipeline (potentially the most expensive component) will be idle

The graphics pipeline might really be software running on the host CPU In which case, the most time consuming operation is likely to be the processing of the millions of pixels that must be rendered For the rest of this discussion, we assume that there is some dedicated graphics hardware for the graphics subsystem

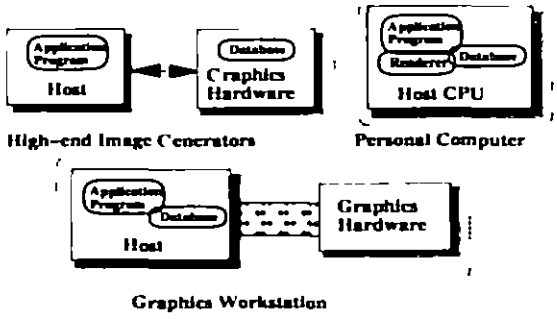


FIGURE 5 Host Graphics Organizations

Consider time for data transfer in addition to pipeline computation

The application may itself be multiprocessed and running on one or more CPUs The host and the graphics pipeline may be tightly connected, sharing a high speed system bus and possibly even access to host memory Bandwidth between the host and graphics subsystem is the first place a bottleneck can occur in getting data to the graphics subsystem Host-graphics bandwidth ranges from approximately 100MBytes/sec with PCI and 500MBytes/sec with Intel's Advanced Graphics Port to the scalable 1 6GBytes per connection of the Origin™ system Bringing paged geometry and texture data off disk may be another front-end bottleneck Disk paging rates range from about 4MBytes/sec at the low end to scalable 90MBytes/sec per channel on Origin Fibre Channel

The first stage of the rendering pipeline is traversal of the database and transmission of the current rendering data on to the rest of the graphics pipeline In theory, the entire rendering database, or *scene graph* must be traversed in some fashion for every frame because both scene content and viewer position are dynamic Because of this, there are three major parts of the database traversal stage processing to determine current viewing parameters (usually part of the main application), determining which parts of the scene graph are contained within the viewing frustum (culling), and the actual drawing traversal that issues rendering commands for the visible

parts of the database These components form a traversal pipeline of three stages Application, Cull and Draw

FIGURE 6 Application Traversal Process Pipeline

Possibilities for the application processes are discussed further in Figure 6 This section will be focussing on the drawing traversal stage of the applica tion

Some graphics architectures impose special requirements on the drawing traversal task such as requiring that the geometry be presented in sorted order from front to back or requiring that data be presented in large, specially formatted chunks as display lists

There are three main types of database drawing traversal

immediate mode
display list mode
retained data

Immediate Mode Drawing Traversal

In the first two, the rendering database lives in main memory For immediate mode rendering, the database is actually shared with the main application on the host as shown in Figure 7 The application is responsible for traversing the database and sending geometry directly to the graphics pipeline This mode is the most memory efficient and flexible for dynamic geometry However, the application is directly responsible for the low level communication with the graphics subsystem Some systems offer a batch method of immediate-mode transfer for large amounts of vertex data that can use more efficient transfer methods such as DMA OpenGL vertex arrays offer one such potential interface for such support

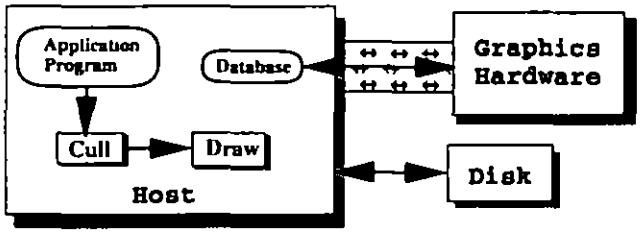


FIGURE 7 Architecture with Shared Database

Display List Traversal

In display-list mode, pieces of the database are compiled into static chunks that can then be sent to the graphics pipe In this case, the display list is a separate copy of the database that can be stored in main memory in a form optimized for feeding the rest of the pipeline The database traversal task is to hand the correct chunks to the graphics pipeline These display lists can usually be edited or re-created easily for some additional performance cost For both of these types of drawing traversals it is essential that the application be using the fastest possible API for communication with the graphics subsystem An inefficient host-graphics interface for such operations as issuing polygons and vertices could leave the rest of the graphics pipeline starved for data

Use the fastest interface to communicate with the graphics pipeline

There is potentially a significant amount of data that must be transferred to the graphics pipeline every frame If we consider just a 5K triangle frame, that corresponds to

(5000 tris) * (3 vertices/tri * (8 floats/vertex) * (4bytes/float)) = 480KBytes

→ 28 8 MBytes/sec for a 60fps update rate

for just the raw geometric data The size of geometric data can be reduced through the use of primitives that share vertices, such as triangle strips or through the use of high-level primitives, such as surfaces, that are expanded in the graphics pipeline (this is discussed further in Section 9) In addition to geometric data, there may also be image data, such as texture maps It is unlikely that the data for even a single frame will fit a CPU cache so it is important to know the rates that this data can be retrieved from main memory It is also desirable to free up the CPU by having the graphics subsystem pull data directly out of main memory For highly interactive and dynamic applications, it is important to have good performance on transfers of small amounts of data to the graphics subsystems since many small objects may be changing on a per-frame basis

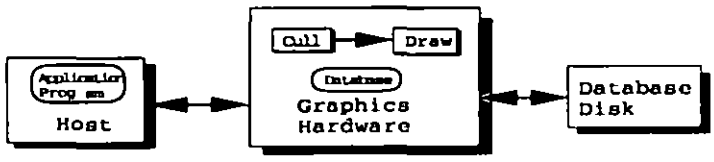


FIGURE 8 Architecture with Retained Data

Retained Database Traversal

If the database and display list is stored in the graphics pipeline itself as shown in Figure 8 separate from main memory, it is a retained display list Retained display lists are traversed only by the graphics pipeline and are required if there is very low bandwidth between the host and the graphics subsystem The application only sends small database edits and updates (such as the new viewing parameters and a few matrices) to the graphics pipe on a per-frame basis Pieces of the database may be paged directly off local disks Retained mode offers much less flexibility and power over editing the database, but can remove the possible bandwidth bottleneck at the head of the graphics pipeline

The use of retained databases can enable additional processing of the total database by the graphics subsystem For example partitioning the database may be done in order to implement sophisticated optimization and rendering techniques One common example is the separation of static from moving objects for the implementation of algorithms requiring sorting The cost may be additional loss of power and control over the database due to limitations on database construction such as the number of moving objects allowed in a frame

4 3 Graphics Subsystems

The second two stages of the rendering pipeline Figure 3 are commonly called *The Geometry Subsystem* and *The Raster Subsystem*, respectively

The Geometry Subsystem

The geometry subsystem operates on the geometric primitives (surfaces polygons, lines points) The actual operations are usually per vertex operations The basic set of operations and estimated computational complexity includes [Foley90] modeling transformation of the vertices and normals from eye-space to world space per-vertex lighting calculations, viewing projection, clipping, and mapping to screen coordinates Of these, the lighting calculations are the most costly A minimal lighting model typically includes emissive ambient diffuse and specular illumination for infinite lights and viewer The basic equation that must be evaluated for each color component (R G, and B) is [OpenGL93]

```
RGBemissive_mat +
RGBambient_model*RGBambient_mat +
RGBambient_mat*RGBambient_light +
RGBdiffuse_mat*RGBdiffuse_light * (light_vector normal)
```

Specular illumination adds an additional term (exponent can be approximated with table lookup)

$$RGBspecular\_light * RGBspecular\_mat * (half\_angle \cdot normal)^{shininess}$$

Much of this computation must be re-computed for additional lights Distance attenuation, local viewing models and local lights add significant computation

A trivial accept/reject clipping step can be inserted before lighting calculations to save expensive lighting calculations on geometry outside the view ing frustum However, if an application can do a coarse cull of the database during traversal, a trivial reject test may be more overhead than benefit Examples of other potential operations that may be computed at this stage include primitive based antialiasing and occlusion detection

This block of floating-point operations is an ideal case for both sub-pipelining and block parallelism For parallelism, knowledge about following issues can help application tuning

MIMD vs SIMD

how streams of primitives are distributed to the processors

whether the output of these processors is to remain separate in parallel streams for the next major stage or re combined into a single output stream for re distribution

The first issue, MIMD vs SIMD, affects how a pipeline handles changes in the primitive stream Such changes might include alterations in primitive type state changes, such as the enabling or disabling of lighting and the occurrence of a triangle that needs to be clipped to the viewing frustum SIMD processors have less overhead in their setup for changes However, since all of the processors must be executing the same code, changes in the stream can significantly degrade processor utilization particularly for highly parallel SIMD systems MIMD processors are flexible in their acceptance of varied input, but can be more complex to setup for given operations, which will include the processing state changes This overhead can also degrade processor utilization However, more processors can be added to balance the cost of this overhead

The distribution of primitives to processors can happen in several ways An obvious scheme is to dole out some fixed number of primitives to processors This scheme also makes it possible to easily re-combine the data for another distribution scheme for the next major stage in the pipeline MIMD processors could also receive entire pieces of general display lists, as might be done for parallel traversal of a retained database

Order rendering to fill the pipeline from back to front

Draw large occluders first

The application can affect the load-balancing of this stage by optimizing the database structure for the distribution mechanism, and controlling changes in the primitive stream

After these floating point operations to light and transform geometry, there are fixed-point operations to calculate slopes for the polygon edges and

additional slopes for z colors, and possibly texture coordinates These calculations are simple in comparison to the floating point operations The more significant characteristic here is the explosion of vertex data into pixel data and how that data is distributed to downstream raster processors

The Raster Subsystem

There is an explosion of both data and processing that is required to rasterize a polygon as individual pixels Typically, these operations include depth comparison gouraud shading color blending logical operations, texture mapping and possibly antialiasing These operations require accessing of various memories, both reading for inputs to comparisons and the blending and texturing operations, and writing of the updated depth and color information and status bits for logical operations In fact the memory accesses can be more of a performance burden than the simple operations being computed Of course, this is not true if complex per pixel shading algorithms such as Phong Shading, are in use For antialiasing methods using super-sampling some of these operations (such as z-buffering) may have to be done for each sub-sample For interpolation of pixel values for antialiasing each pixel may also have to visit the memory of its neighbors Texture interpolation for smoothing the effects of minification and magnification can also cause many memory accesses for each pixel An architecture might choose to keep some memory local to the pixel processor, in which case fill operations that only access local processor memory would probably be faster

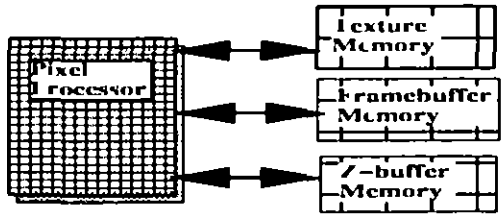


FIGURE 9 Pixel Operations do many Memory Accesses

The vast number of pixel operations that must be done would swamp single-processor architectures but are ideal for wide parallelism The Silicon Graphics InfiniteReality has 80 Image Processors on just one of up two to four raster subsystem boards The following is a simplified example of how parallelism can be achieved in the Raster Subsystem The screen is subdivided into areas for some number of rasterizing engines that take polygons and produce pixels Each rasterizer has a number of pixel processors that are each responsible for a sub area of its parent rasterizing engine and

writes directly into framebuffer memory Thus the Raster Subsystem may have concurrent sub-pipelines

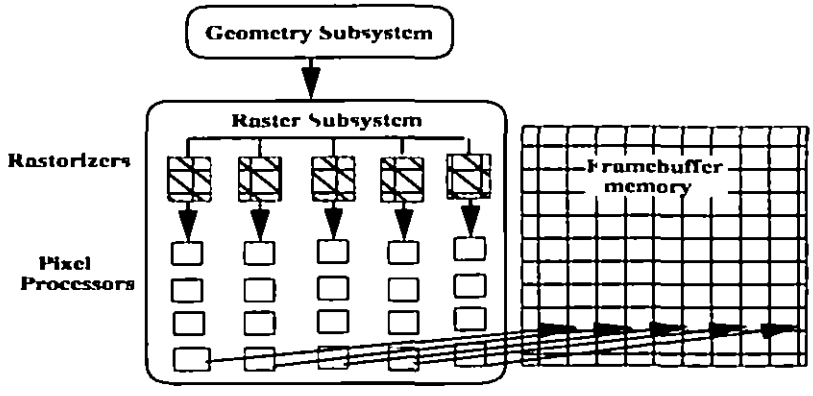


FIGURE 10 Parallelism in the Raster Subsystem

In some architectures such as the Silicon Graphics VGX™, the rasterizers get interleaved vertical spans on the screen and the pixel processors get interleaved pixels within those spans The Silicon Graphics Reality Engine™ and InfiniteReality architectures use similar schemes, but each with more complex interleaving for better load balancing and many more pixel processors[Akeley93][Montrym97]

Benchmark fill operations with different modes and different polygon sizes and orientations

Certain operations that can cause an abort of the writing to a pixel such as a failed depth buffer test can be used to short-circuit further expensive pixel operations If the application can draw front to back or draw large front polygons first a speedup might be realized

Depending on the distribution strategy MIMD processors in this stage might show more benefit from such short-circuit operations The distribution strategy typically employs an interleaved partitioning of the frame buffer This optimizes memory accesses and promotes good processor utilization The possible down side is that most processors will need to see most primitives The complexity in figuring out which primitives go to which processors may cause processors to receive input for which they do no work Because of this overhead, small polygons can have less efficient fill characteristics

Pipeline Connections

The bottleneck of a pipeline may not be one of the actual stages, but in fact one of the buses connecting two stages, or logic associated with it In the graphics subsystem, there may be logic for parsing the data as it is comes off the bus, or distributing the data among multiple downstream receivers

Any connection that must handle a data explosion such as the connection between the Geometry and Raster subsystems is a potential bottleneck The only way to reduce these bottlenecks is to reduce the amount of raw data that must flow through it or to send data that requires less processing The most important connection is the one that connects the graphics pipeline to the host because if that connection is a bottleneck the entire graphics pipeline will be under utilized

The use of FIFO buffers between pipeline stages provides necessary padding that protects a pipeline from the affects of small bottlenecks and smooths the flow of data through the pipeline Large FIFOs at the front of the pipeline and between each of the major stages can effectively prevent a pipeline from backing up through upstream stages and sitting idle while new data is still waiting at the top to be presented This is useful for fill-intensive applications which tend to bottleneck the very last stages in the pipeline However, once a FIFO fills, the upstream stage will back up

Video

The final stage in the frame interval is the time spent waiting for the video scan-out to complete for the new frame to be displayed This period, called a *field* is the time from the first pixel on the screen until the last pixel on the screen is scanned out to video For a 60Hz video refresh rate the time could be as much as 16 7msecs Graphics workstations typically use a dou ble-buffered framebuffer so that for an extra field of latency, the system can achieve frame rates equal to the scan out rate A double-buffered system will toggle between two framebuffers outputting the contents of one framebuffer while the other is receiving rendering results The framebuffers cannot be swapped until the previous video refresh has completed This will force the frame rate of the application to run at a an integer multiple of the video refresh rate In the worst case, if the rendering for one frame completed just after a new video refresh was started, the application could theoretically have to wait for the entire refresh period waiting for an available framebuffer to receive rendering for the next frame

Frame rate is descretized by video rate

The time for video refresh is also the lower bound on possible latency for the application The typical double buffered application will have a minimum of two fields of latency one for drawing the next frame while the current one is being scanned and then a second field for the frame to be scanned This assumes that the frame rate of the application is equal to the field rate of the video In reality, a double-buffered application will have a latency that is at least

$$2 * N * field\_time$$

where N is the number of fields per application frame Techniques for reducing latency are discussed in "Reducing Latency" on page 39

Multiprocessing of Entire Graphics Pipelines

Throw hardware at the problem!

Multiple graphics pipelines parallel synchronized or else pipelined and time interleaved

The multiprocessing methods of parallelism and pipelining can be employed within a graphics pipeline as well as at the level of the full graph ics pipeline Multiple graphics pipelines can be utilized to produce in parallel different blocks of a single frame image, or pipelined to produce the alternating even/odd lines of an interlaced display, or pipelined to produce alternate and interleaved full frames Multiple parallel pipelines producing a single frame have the least inherent latency of these solutions unless another stage is required to composite the images into a framebuffer (as opposed to providing separate video outputs) In addition, unless the area of responsibility of the blocks of the framebuffer can be dynamically determined on a per-frame basis (as with the Silicon Graphics RealityMonster™), some pipelines might have a much greater load than other pipelines producing unpredictable performance This is likely with a complex scene because of the natural compression that happens at the horizon line Multiple parallel graphics pipelines also require some kind of hardware synchronization to make sure that they all display their output fields simultaneously The technique of interleaving in time different output frames from different graphics pipelines adds an extra field of latency for each graphics pipeline in the chain but produces a reliable system load and each output frame is the full image requiring no additional compositing

Commercial Graphics Architectures

It is always fascinating to see the creative ways in which basic concepts can be applied to produce vastly different architectures On traditional image-generators, the main application is actually running on a remote host with a low-bandwidth network connection between the application running on the main CPU and the graphics subsystem The full graphics database resides in the graphics subsystem Developing applications on these machines is a matter of tuning the database to match set performance specifications Until recently, almost all of the graphics processing for PCs has traditionally been done by the host CPU and there has been little or no dedicated graphics hardware Recently, there have been many new developments in this area with dedicated graphics cards developed by independent vendors for general PC buses Some of these cards have memories for textures and even resident databases

Graphics workstations fall between these two extremes They traditionally have separate processors that make up a dedicated graphics subsystem They may also have multiple host CPUs Some workstations, such as Sili-

con Graphics have a tight coupling between the CPU and graphics subsystems with system software compilers and libraries However there are also independent vendors such as Evans & Sutherland producing both high and low end graphics boards for general workstations

The following are brief examples of a few different past and current graphics architectures with different price/performance characteristics The choices in architecture shift with considerations forced by technology but the constant trend is the steady increase in performance given by the ever growing demands of the market at all levels

Apollo DN10000[Voorhies89] — A complete graphics workstation RISC concepts were applied to produce a RISC graphics capable CPU that could be combined in a multiprocessor configuration All traversal and geometry processing are done on the CPU(s)

SGI Lxtreme[Harrell93] — desktop workstation with 500K triangles/sec and 80MPixels gouraud zbuffered The geometry subsystem is 8 SIMD parallel processors raster subsystem is 2 parallel processing blocks Hyper pipelining in the raster subsystem is used to increase performance of pixel generation Texture mapping done in software on the host

Silicon Graphics high end graphics workstations — three generations of graphics pipelines native to a multiprocessor host

GTX[Akeley89] — has a highly parallel Geometry Subsystem with five stages and a raster subsystem that includes parallelized stages for scan conversion and pixel processing

VGX[Akeley90] — achieves high polygon rates (first workstation to do 1 Million triangles/sec) with a simpler 3 stage geometry subsystem where two of the stages use four SIMD processors each Data was combined at the output of the geometry subsystem for re distribution to the raster subsytems which were MIMD parallelized and extendable Texture mapping is supported at reduced fill rates VGX Skywriter supported multiple graphics pipelines in one (large) box that could act as independent parallel pipelines or could be used together as frame interleaved pipelines to effectively double throughput (but with additional latency)

RealityEngine[Akeley93] — has a simpler geometry subsystem that is 12 MIMD processors wide and has an even more extendable raster subsystem Up to three independent pipelines may exist in a single box High polygon rates (1 2M 1 8M on RE2) were in balance with high fill rates for complex fill algorithms including tri linear texture mapping and real time full scene antialiasing Also supports per window stereo

InfiniteReality[Montrym97] — custom SIMD processors in a scalable geometry subsystem produced 10millions textured triangles/sec for 4 GEs Scalable raster subsystem produces 600MPixels/sec for Antialiased textured zbuffered etc pixels Massive multipipe configurations are possible (up to 8 at the time of this writing are supported)

Kubota Denali(93) — scalable graphics subsystem that sits on the main DEC Alpha system bus Has main stages of 3 stage Geometry Transformation Module and Framebuffer Module(SIMD) both are extendable for greater performance Has low to high end configurations with 200K and 1 2M triangles/sec respectively and supports texture mapping at reduced fill rates

Sun Leo[Deering93] — desktop workstation capable of 210K 100 pixel triangles/sec Geometry subsystem uses four parallel custom floating point processors and the raster subsystem uses five parallel processors Leo is VRAM access limited and thus fill limited even for 100 pixel triangles Supports per window stereo

Evans & Sutherland Image Generators — use a remote networked host and retained database traversal They have special support for many visual simulation features including special terrain traversal and rendering algorithms with dynamic Level of Detail (LOD) landing lights and various atmospheric affects The ESIG4000(high end) has special support for the placement of LOD geometry on LOD terrain and distortion correction for projection onto curved display surfaces Restrictions on databases include the databases must be built with special modeling tools and use separating plane construction for peak performance and an associated limited number of moving models allowed and animated forms of dynamic objects

PixelPlanes[Fuchs89] explores the use of massively parallel processors for the rendering task A MIMD array of geometry processors sorts polygons into screen based region buffers (128x128 pixels) and SIMD pixel processors apply pixel based rendering algorithms such as Phong shading These sophisticated shading calculations are deferred until an end of frame point after zbuffering so they only have to be done once per pixel and are not affected by depth complexity All geometry processing must be done before the deferred shading pass Screen based allocation does not provide good load balancing

PixelFlow[Molnar93] improves this load balancing It is an image composition architecture where rendering of primitives is distributed over an extendable massively parallel array of complete renderers — each able to draw to the entire frame buffer but still rasterizes in 128x128 pixel patches As in PixelPlanes deferred shading is applied after the renderer images are composited PixelFlow can run in either a retained database mode or in an immediate mode where it is hosted by a parallel computer with processors directly connected to the renderers

5 *Optimizing Performance of a Graphics Pipeline*

Tuning of both the application and database is an essential part of the development process on a commercial graphics workstation Higher-level hardware-savvy software tools like IRIS Performer greatly ease this burden but by no means remove it Traditionally, both high-end image generators and low-end PC graphics platforms have been very restrictive in the type of application that can be supported and the scene content that can be rendered at a given rate Graphics workstations offer

- a wide range of graphics subsystems tightly coupled with today s fastest CPUs

- high bandwidth connections between the main CPU and the graphics subsystem
- scalable (binary compatible) product lines and graphics standards for portability to different architectures and hopefully future architectures
- native optimized standard rendering libraries such as OpenGL
- the flexibility of being able to make performance trade offs to optimize performance maximize scene content and minimize cost — careful tuning can yield tremendous results resulting in a flexible low cost solution with high scene content
- sophisticated development environments

Tune early and tune often it only gets harder as your app gets larger

The down side of these features is that the tuning process is not only essential, it can be complex Tuning an application to be performance-portable to different architectures is additionally complex Unfortunately, tuning is one of those tasks that is often put off until the point of crisis The *Law of Diminishing Returns* definitely applies here for an application designed with performance in mind it is typically fairly easy to achieve about 60% of expected optimal performance A bit more work can get you to 75% or 80% Then it starts to get much more difficult because of the pure complexity of having so many system parameters interacting and affecting performance The key is in identifying and isolating the current problems The goal is a balanced pipeline where no one stage is an overwhelming bottleneck

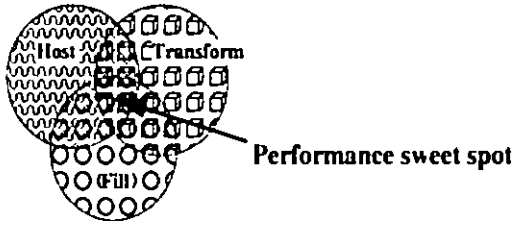


FIGURE 11 A Balanced Pipeline

The focus of this section is the development of the following basic tuning strategy

1 Design for performance
2 Estimate expected performance
3 Measure and evaluate current performance
4 Isolate performance problems
5 Balance operations
6 Repeat

5 1 Design For Performance

The full system pipeline should be kept in mind when designing the application Graphics features should be chosen to balance the pipeline and

careful estimations of expected performance for target databases should be made during the design phase as well as the tuning phase

Combinations of rendering features should be chosen to produce a balanced pipeline An advantage of graphics workstations is the power to make trade-offs to maximize both performance and scene quality for a given application If, for example, a complex lighting feature is required that will bottleneck the geometry subsystem then possibly a more interesting fill algorithm could be used to both require less polygons being lit and achieve overall higher scene quality

Be aware of features that use multi-pass algorithms because pipelines are usually balanced with one pass through each stage There are many sophisticated multi-pass algorithms incorporating such techniques as texture mapping Phong-shading accumulation antialiasing, etc , that produce special effects for high-quality images Multi-pass algorithms make efficient use of generalized hardware and so are an example of the powerful available trade-off options between performance and image quality Because they use simple and generalized hardware operations (such as texture mapping and copy), they have been benefited by improvements to such hardware and are rapidly becoming far more efficient They should be used when their effect is desired and their performance impact should be budgeted Useful examples of multi-pass techniques are discussed in section "Multipass texture techniques" on page 46

Be Multiprocessed from the start

The application should also be designed with multiprocessing in mind since this is very hard to add after-the-fact Large tasks that can be run on separate processors (preferably with minimal synchronization and sharing of data) should be identified For ease of debugging portability, and tuning (discussed further in Section 7) the application should support both a single process mode and a mode where all tasks are forced into separate processes

The tasks also need to be able to non-invasively monitor their own performance, and need to be designed so that they will support measurements and experiments that will need to be done later for tuning The rendering task (discussed later in this section) must send data to the graphics pipeline in a form that will maximize pipeline efficiency Overhead in renderer operations should be carefully measured and amortized over on-going drawing operations

5 2 Estimating Performance for a Pipeline

The first step both in designing a real-time application and tuning an existing application, is to assess the requirements for image quality and predict the time required to produce that image quality If there is a large gap

between these calculations trade-offs may have to be made in the application to balance scene content with performance to produce the most compelling result

Making careful performance estimations greatly enhances your understanding of the system architecture If the target machine (or similar machine) is available then this should be done in tandem with the analysis of current application performance and the comparison to small benchmarks until the measurements and estimations agree

Compute expected performance for each stage of the pipeline on the total scene

As should not be surprising by this time estimating performance of an application for a pipeline is much more than looking at peak quoted numbers for a machine and polygon totals for a database The following are basic steps for estimating performance

1. Define the contents of a worst case frame including number of polygons and their types number of graphics modes and changes and average polygon sizes
2. Identify the major stages of the graphics pipeline
3. For each major stage identify the parts of the frame that are significant for that stage
4. Estimate the time that these frame components will spend in each stage of the pipeline (if possible verify with small benchmarks)
5. Sum the maximum stage times computed in (4) for a very pessimistic estimation
6. When the drawing order can be predicted (such as for screen clear which may be expensive in the fill stage and will always come first) more optimistic estimations can be made by assuming that time spent in upstream stages for later drawing will be concurrent with the downstream work and thus achieved for free

First consider the geometry subsystem the significant operations might be triangles (with better rates for meshed triangles), lighting operations, clipping operations mode changes, and matrix transformations Given this information, one can compile the following type of information

- Triangles percentage of triangles in meshes of different lengths
- Graphics modes percentage of the triangles that are lit
- Mode changes number of texture changes per frame
- Viewing Matrix transformations number per frame
- Clipping percentage of triangles that are trivial accept are trivial reject and those that intersect the viewing frustum

Given the scene characteristics, one should first write small benchmarks to get geometry subsystem performance statistics on individual components Then, write an additional benchmark that mimics the geometry characteristics of a frame to evaluate the interactions of those components

We then similarly examine the raster subsystem We first need to know the relevant frame information

- Depth complexity of the frame and target resolution
- Number of triangles in different fill modes and a few typical sizes
- Number of raster mode changes
- Time for screen clear

Again, if possible benchmarks should be written to verify the fill rates of polygons and the cost of raster mode changes From these estimates, one can make a best guess about the amount of time that will be spent in the raster subsystem

We can now make coarse-grained and fine-grained estimations of frame time An extremely pessimistic approach would be to simply add the bottleneck times for the geometry subsystem and the raster subsystem However, if there is a sufficient FIFO between the geometry and raster subsystems, much of the operations in the geometry subsystem should overlap with the raster operations Assuming this, a more optimistic coarse-grained estimation would be to sum the amount of time spend in the raster subsystem and the amount of time beyond that required by the geometry subsystem A fine-grain approach would be to consider the bottlenecks for different types of drawing Identify the parts of the scene that are likely to be fill-limited and those that are likely to be transform-limited Then sum the bottleneck times for each

5 3 Measuring Performance and Writing Benchmarks

Write benchmarks to target individual pipeline stages

Being able to make good performance measurements and write good benchmarks is essential to getting that last 20% of performance To achieve good timing measurements, do the following

1. Take measurements on a quiet system Graphics workstations have fancy development environments so care must be taken that background processes such as a graphical clock ticking off seconds or a graphical performance monitor etc are not disrupting timing
2. Use a high resolution clock and make measurements over a period of time that is at least 100x the clock resolution
3. If there are only very low resolution timers (less than 1millisecond) then to accurately time a frame pick a static frame (freeze matrix transformations) run in single buffered mode and time the repeated drawing of that frame
4. Make sure that the benchmark frame is repeatable so that you can return to this exact frame to compare the affects of changes
5. Make sure that pipeline FIFOs are empty before starting timing and then before checking the time at the end of drawing When using OpenGL one should call `glFinish()` before checking the clock
6. Verify that you can rerun the test and get consistent timings

A generally good technique for writing benchmarks is to always start with one that can achieve a known peak performance point for the machine If you are writing a benchmark that will do drawing of triangles, start with one that can achieve the peak triangle transform rate This way if a bench mark seems to be giving confusing results you can simplify it to reproduce the known result and then slowly add back in the pieces to understand their effect

When writing benchmarks, separate the timings for operations in an indi vidual stage from benchmarks that time interactions in several stages For example to benchmark the time polygons will spend in the geometry sub-system make sure that the polygons are not actually being limited by the raster subsystem One simple trick for this is to draw the polygons as 1-pixel polygons Another might be to enable some mode that will cause a very fast rejection of polygon or pixels after the geometry subsystem How-ever, it is important to write both benchmarks that time individual opera-tions in each stage, and those that mimic interactions that you expect to happen in your application

5 4 Finding the Bottlenecks

Over the course of drawing a frame, there will likely be many different bot-tlenecks If you first clear the screen and draw background polygons you will start out fill-limited Then, as other drawing happens the bottleneck will move up and down the pipeline (hopefully not residing at the host) Without special tools bottlenecks can be found only by creative experi-mentation The basic strategy is to isolate the most overwhelming bottle-neck for a frame and then try to minimize it without creating a worse one elsewhere

One way to isolate bottlenecks is by eliminating work at specific stages of the pipeline and then check to see if there is a significant improvement in performance To test for a geometry subsystem bottleneck you might force off lighting calculations or normalization of vertex normals To test for a fill bottleneck disable complex fill modes (z buffering gouraud shading texturing) or simply shrink the window size However beware of second-ary affects that can confuse the results For example if the application adjusts what it draws based on the smaller window, the results from just shrinking the window without disabling that functionality will be meaning-less Some stages are simply very hard to isolate One such example is the clipping stage However if the application is culling the database to the frustum, you can test for an extreme clipping bottleneck by simply pushing out the viewing frustum to include all of the geometry

6 Tuning the Application

Applications usually plan on pushing graphics past their limits If the rendering traversal is part of the application then this traversal must be optimized so that it keeps the graphics subsystem busy On a multiprocessing system, operations such as scene management and paging and formatting of data can be moved out of the renderer and into other processes preferably running on other CPUs A key part of real-time rendering is *load management*, providing a graceful response when the graphics subsystem becomes overloaded, which is discussed later in Section 7

6 1 Tuning the Renderer

There is no escape from writing efficient code in the renderer Immediate mode drawing loops are the most important parts since code in those loops are executed thousands of times per frame For peak performance from these loops, one should do the following

- Minimize the loop overhead and decision logic in the polygon loops Unroll per vertex code and duplicate code as opposed to using per vertex if tests
- Use a flat data structure for the draw traversal you want to minimize the number of memory pages you need to touch in a given loop
- Disassemble code to examine loop overhead (and to check that the compiler is doing what you expect)

Display list rendering requires less optimization because it does not require the tight loops for rendering individual polygons However this is at the cost of more memory usage for storing the display list and less flexibility in being able to edit the geometry in the list for dynamic objects The extra memory required by display lists can be quite significant because there can be no vertex sharing in display lists This can restrict the number of objects you can hold in memory and will also slow the time to page in new objects if the graphics display lists must be re-created Additionally, display lists may need to be of a certain minimum size to be handled efficiently by the system If there are many small moving objects in the scene, the result will be many small display lists If you have the choice, given the option between immediate mode rendering and database paging you might choose to use at least some immediate mode, particularly for dynamic objects

Don t let the host be the bottleneck

IRIS Performer uses a fairly aggressive technique for achieving high-performance immediate-mode rendering Data structures for geometry enforce the use of efficient drawing primitives Geometry is grouped into sets by type and attribute bindings (use of per-vertex or per-polygon colors, normals, and texture coordinates) For each combination of primitive and attribute binding, there is a specialized routine with a tight loop to draw the

geometry in that set The result is several hundred such routines but the use of macros makes the code easy to generate and maintain IRIS Performer also provides an optimized display list mode that is actually an immediate mode display list and shares the application copy of data instead of copying off a separate uneditable copy This is discussed in [Rohlf94] and [PFPG97] Host rendering optimization techniques are also discussed in detail in [GLPT192]

Multiprocessing

Use coarse grained multiprocessing first

Multiprocessing can be used to allow the renderer to devote its time issuing graphics calls while other tasks, such as scene and load management can be placed into other processes There are several large tasks that are obvious candidates for such course-grained multiprocessing

- the real time application — processes inputs from IO calculates new viewing parameters positional parameters for objects and parameters for dynamic geometry
- scene management — culling out parts of the scene graph that are not in the viewing frustum calculating LOD information generating a display list for the rendering traversal
- dynamic editing of geometric data
- IO handling — polling external devices database paging
- intersection traversals for collision detection
- complex simulations for various vehicles
- computations for light points and calligraphic lights

A combination of pipelining and parallelism can be used to get the right throughput/latency trade-off for your application and the target machine IRIS Performer provides a process pipeline

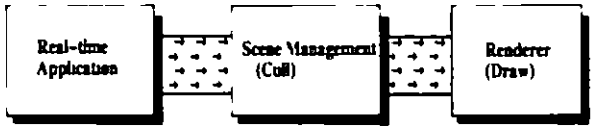


FIGURE 12. IRIS Performer Process Pipeline

This process pipeline, described in [Rohlf94] is re-configurable to allow

- a pipeline where the cull and draw are parallel processes (app >cull_draw)
- a model where the cull and draw are performed by a single process that culls and renders simultaneously (app >cull&draw)
- a minimal latency model where all tasks are performed by a single process (app&cull&draw)

Multiprocessing also allows additional tasks to be done that will make the rendering task more efficient, such as

generating a per frame optimized display list for the rendering task so that the drawing traversal does not need to traverse the original database

sorting geometry by mode to minimize mode changes

host backface removal (and removal of backfaced objects) to save additional host bandwidth

flattening of dynamic transformations over objects of only one or two polygons

7 Real-Time

It is important to identify which tasks must be real-time and which can run asynchronously and extend beyond frame boundaries Real-time tasks are those that must happen within a fixed interval of time, and severe consequences will result if the task extends beyond its frame The main application, cull, and draw tasks are all real-time tasks However, it might not be so traumatic if for example, some of the collision results are a frame late The polling of external control devices should probably be done in a separate, asynchronous process — if those results are late, extrapolation from previous results is probably better than waiting

Don t poll devices from real time processes

7 1 Managing System Resources for Real-Time

One type of organization is to put the rendering process on its own processor, isolated from other system activity and synchronization with other tasks This is the organization used in IRIS Performer[Rolf94] To do this, the rendering process should also have its own copy of data to minimize synchronization and conflict over pages with other processors On a general-purpose workstation, one CPU will need to be running basic system tasks and the scheduler Additionally, a distinction should be made between tasks that must be real-time (happen reliably at fixed intervals), and those processes that may extend past frame boundaries in generating new results Non-real-time tasks can be given lower priorities and share processors perhaps even the system CPU

Time consuming tasks can be placed in separate asynchronous processes to keep them from affecting the rendering frame rate For example the dynamic morphing of terrain LODs and the database paging do not need to happen on a per-frame basis or at real-time rates IRIS Performer places these operations in special processes that reflect their performance characteristics COMPUTE and DBASE processes, respectively The results are merged into the active scene graph when the COMPUTE and DBASE processes are done There is a delay between the time when an asynchronous process captures the view point to the time when the new data is paged in or generated, during which the view point can be actively moving There-

fore the interest area needs to be enlarged to compensate on the movement of viewpoint In the ASD operation as will be described in the terrain LOD section the culling frustum is enlarged In normal database paging operation the paging region also needs to be enlarged The enlarging amount is determined based on the maximum speed of the viewpoint the paging/generation rate of the data the merging cost, and various other potential application details

7 2 Synchronization

Synchronize to video and hopefully nothing else

Synchronization is usually something to be avoided as synchronization points are places where performance can be lost However it is necessary that there is a consistent system wide heartbeat that defined the actual start of each frame which all processes can potentially synchronize for such things as grabbing a new viewpoint and starting computation for the next frame This heartbeat will need to be matched to the graphics pipeline so that rendering can begin at the actual start of the frame This requires that this heartbeat needs to be tied (preferably in hardware) to the start of vertical retrace period when the new framebuffer starts to be displayed and the old framebuffer is ready to receive new data

Visual simulation applications frequently have multiple viewpoints that must be rendered and may require multiple graphics pipelines or full systems If it is desired that these channels display synchronously then the graphics output must be synchronized, as well as the potential host applications driving them in the case of separate machines There is typically some mechanism to synchronize multiple video signals However double-buffered graphics pipelines must also swap buffers during the same video refresh period for output frames to actually be synchronized and not be off by a field This can be done reasonably well from the front end via a high-speed network such as SCRAMnet as was done in the CAVE environment[CN93] or better with special external signals, as is done on the RealityEngine™

7 3 Real-Time on a Workstation

Graphics workstations are attractive development platforms because they have rich and user-friendly development environments They are not traditionally known for their real-time environments However extensions can be made to the basic operating system to support a real-time mode as was done with the Silicon Graphics REACT™ extensions to IRIX™ The REACT extensions are really ways to push UNIX aside for real-time operation It guarantees interrupt response time on a properly configured system and enables the user to have control of the scheduling of specified processors and therefore be exempt from all UNIX overhead REACT also

includes a real-time frame scheduler that can be used on the real-time processors Finally REACT offers time-stamps on system and user operations for real time system performance feedback

Running an application in performance-mode might be quite different from running it in development mode Most obviously, a real-time application needs fast timers to be able to monitor its performance for load management as well as having accurate time-based animations and events A real-time application also needs to be guaranteed worst case behavior for basic system functions such as interrupt response time It also needs to have control over how it is scheduled with other processes on the system, and how its memory is managed In addition, the main application needs to synchronize frame boundaries of various tasks with the graphics subsystem

8 Visual Simulation Application Techniques for Real-Time Graphics

Once the fundamentals are mastered, there is a strong tradition of sophisticated application and database management techniques from visual simulation that map well to modern graphics architectures for enabling the real-time rendering of very complex databases and the production of images with high realism and rich scene content at steady real time frame rates The most obvious technique would be to only render very simple scenes such that meeting the goal frame rate is not a challenge However, this does not make good use of the rendering hardware and does not produce an application that can be moved to other hardware without re-tuning of the database At some point all techniques for optimization and achieving real-time involve all parts of the system database and application This section discusses application techniques that are applicable to most visual simulation databases

Mutliprocessing enablles sophisticated application scene and load management

8 1 Scene Management

The amount of the database that is visible and the resolution at which details can be discerned puts one useful upper limit on the requirements of rendering a scene in a database

Visibility

The first task is to not draw objects that are not within the viewing frustum Particularly in out-the-window visual simulation applications, this will remove most of the database right away This gets us to only drawing our visible streets from our city (or universe) database when at an intersection

The next level of sophistication in this technique is to remove objects or polygons that are occluded by other geometry This technique is useful in dense scenes such as downtown streets or indoor scenes such as building or ship interiors For interiors it is often possible to pre-compute information about potentially visible parts of the database from different points of view and the windows or portals that limit visibility This information can be used for very aggressive occlusion culling of the like in [Luebke95][Teller91][Airey90] This style of occlusion culling requires a preprocessing stage but produces very fast run-time behavior and has the additional advantage that invisible objects are never sent to the graphics pipeline but are eliminated wholesale These are good characteristics for steady real time frame rates More general algorithms not requiring database processing exist at the cost of some use of the graphics hardware for computing the occlusion information at run-time for finding potential occluders or computing at low resolution that a bounding volume of an object is not visible and therefore the object is not visible[Zhang97]

However in all of these algorithms, if a given object has any portion of it visible the entire object will likely be drawn In natural scenes, this is often the case Some examples include looking down the aisle of a plane or down a street of buildings, where there are many layers of objects and polygons all partially occluded For these types of scenes, hardware assistance to eliminate the rendering of invisible pixels is desirable

Level of Detail

Visibility techniques keep you from drawing entire objects that are truly impossible to see However objects or polygons that are so small as to not cover even a single pixel on the screen are effectively invisible The next step in scene management is controlling the resolution, or level of detail (LOD) at which objects are rendered The generation of different resolution forms of a given object can be a very expensive process and automatic algorithms are still in the research phase However off-line processing can produce different representations of objects paired with criteria expected by the application for when each representation is appropriate The different representations can consider numbers of polygons used as well as numbers of textures and other state attributes to produce low resolutions objects that are simple for the rendering hardware

Common LOD selection criteria include distance from the eye angle from the gaze direction screen area of object and number of visible pixels in the object Some of the LOD criteria itself can be expensive to compute at run-time and can benefit from hardware acceleration or multiprocessing The criteria used for LOD selection must be carefully balanced against the artifacts that might be caused at run-time from visible transitions between two

LODs For example, screen area is a good criterion for considering the potential raster load of an object However, depending on the shape of the object and how the screen area is computed, the object might wildly oscil late in representations even when it is very visible

The transition between two LODs, if simply a direct switch is likely to be very visible unless the switch happens at a very conservative time so that the higher resolution us used beyond its range of usefulness However, the LOD transition can apply more sophisticated algorithms to span the transi tion over some number of frames or distance changed and in this transition range rendering some intermediate form given the two LODs for each end of the range

Fade or morph for smooth LOD transitions

Fading using transparency to blend between two arbitrary representations of a model produces a very effective result This techniques works on arbi trary objects and does not require any special modeling or corrolation between different LODs In the transition range both representations of the object are drawn with complementary transparencies that sum to one As an object nears to or backs from the eyepoint, one LOD of the object fades in to become the new chosen LOD and the previous chosen LOD fades out Since the objects are likely to be colored and are drawn together in place, they require special handling to get the proper transparent combining Traditional blending would require sorting the polygons which would be expensive at best but likely impossible Any antialiasing algorithm that forces front-to-back rendering would also conflict with this technique The supersampling style of antialiasing on the InfiniteReality allows for a screen-door style of transparency that easily enables this simple fading LOD technique However, fading is not ideal Though it produces an effec tive transition, fading produces an expensive transition as temporarily more geometry is being rendered and done so in an expensive fill mode The transition could temporarily worsen the very performance that we are trying to improve with the LOD technique

A very effective LOD transition alternative to fading is *morphing* Morphing is applied to the vertices of the object Described in the sense of transitioning from a lower LOD to a higher LOD, the vertices of a higher LOD exist simultaneously with vertices of the lower LOD and are coplanar with the lower LOD polygons The higher LOD vertices then are gradually transformed over the transition frames to their target position With the proper constraints on the vertices between LODs, this method produces results of high image quality and does not produce the performance instability of fading The constraints on the vertices are to minimize potential distortion artifacts that can arise during the morphing transition For generalized objects this is challenging but much success has been achieved in

using morphing to achieve continuous LOD in terrain the most significant object in most flight and ground-based visual simulation scenes

Terrain LOD

Geometric morphing for LOD as in [Hop96], eliminates the need to temporarily render additional triangles and interacts better with depth buffer hidden surface elimination at the expense of increased computation on the general purpose processors (CPUs) to compute the geometric morph parameters

Much of the work on level of detail reduction has concentrated on the creation and coherent transition between LODs where the whole object is rep resented in one LOD[9, 10] When the object being rendered is very large and the viewpoint is such that some portions of the object are much closer to the viewer than others, there is no single level of detail optimal for the entire object Instead it is desirable to render the object with a smooth variation of level of detail, allowing nearer portions to be represented accu rately, and farther portions to be represented efficiently The need for spatially varying level of detail is particularly acute when rendering terrain which may range from two feet to fifty miles from the viewpoint

Terrain is typically represented and displayed using some kind of triangle mesh There are two kinds of meshes 1) the uniform mesh where vertices fall on a regular 2D or 3D grid and 2) Triangulated Irregular Networks (TINs) where triangles in the mesh are arbitrarily shaped and vertices are free to land exactly on real data points at any level of detail The terrain meshes are normally generated from raw uniformly sampled elevation data The trade off between these two meshing styles is construction speed vs triangle efficiency While uniform meshes can be generated from raw data easily, TIN can represent the same terrain with much fewer triangles TIN naturally satisfies tessellation constraints such as preserving ridge lines grid meshes force many triangles to set proper accuracy

Early attempts to support multiple LOD terrain modeled the surface as separate patches, each rendered at a single level of detail This approach results in physical discontinuity between neighboring patches that are in different levels of detail which are visible as obvious ' walls and cracks" Smooth multi-resolution meshing for large surfaces especially terrain, is currently a very active research topic Lindstrom et al [Lin96] addressed this issue by switching different levels of detail which are uniform meshes at a much finer granularity, resulting in far less visual distraction However, like all methods of uniform sampling their algorithm requires far more triangles than are needed by a non-uniform tessellation of similar accuracy The advantage is that the uniform meshing produces very long triangle

strips Hoppe presented a progressive meshing technique for both view independent and view dependent cases [Hoppe96][Hoppe97] De Floriani et al introduced Hierarchical triangulation concept[Floriani95]

The type of evaluation and tessellation is just part of the terrain LOD problem There must be a real-time framework for managing the evaluation at run-time IRIS Performer2 2 presents a full general framework for organizing large amounts of terrain data in the Active Surface Definition (ASD) data structure for various types of terrain LOD evaluation ASD stores multiple LODs in a subdivision-like hierarchical structure The ASD evalua tion traverses the structure selecting triangles from different LODs to render the portion of the terrain within the culling frustum In general, ter rain that is closer to the viewer is represented with more finely tessellated triangle meshing

Avoid popping and cracking with smooth terrain LOD

The ASD structure eliminates cracks and T-vertices by imposing constraints on the relationship between parent and child morphing shown in Figure 13 Triangle replacement shares edges and the new edges completely replace the original edge for both triangles that shared it, even if only one triangle needed to be rendered at the higher LOD, as shown in B Final positions of the vertices of the new edges are shown in C

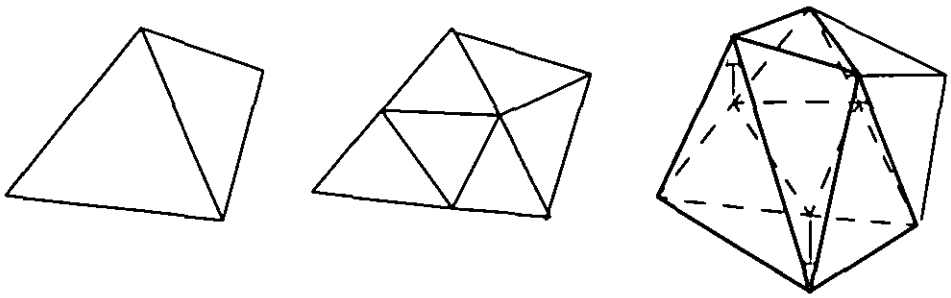


FIGURE 13 Terrain LOD Morphing

Detailed geometry are morphed in or out gradually and cause very small/ invisible visual distractions The detail evaluation can be based on range to view point, which is the default function, or any other factors such as pro jected on screen errors, viewing angles, or dynamic events

ASD supports geometry morphing tight view frustum culling collision detection, as well as real-time database paging Cultural features such as trees and light points can be attached actively to the changing terrain The evaluation of meshes can be done asynchronously, and therefore avoiding impacting the performance of the main IRIS Performer pipeline ASD pro

vides good load control by allowing user to tune the amount of geometry to be rendered

8 2 Load Management

Scene management techniques have the most influence in approaching real-time frame rates However for achieving truly fixed frame rates some type of additional control is usually needed High-end image generators have frame control built into the graphics subsystem so that they can simply halt drawing at the end of a frame time This can produce an unattractive result but perhaps one that is less disturbing than a wild frame rate Getting a graphics subsystem to stop based on a command sent from the host and placed in a long FIFO can be a problem If the graphics subsystem does not have its own mechanism for managing frame rate control (as currently only high-end image-generators do) then the host will have to do it This means leaving generous margins of safety to account for dynamic changes in graphics load and tuning the database to put an upper bound on worst case scenes However, some method of load management will also be required Load management requires at minimum a means for dynamic control of the rendering load of the database and the ability to automatically decrease this load when desired Next, there must be a basis for deciding a goal for system load This can be feedback from previous frames on how long rendering has been taking and comparing that to the goal frame rate of the system As load increases the system can detect that it is under stress and begin to reduce load Some systems also incorporate some kind of predictive information[Funk93] to help compute how long the next frame might take to avoid an overload condition Computation for load management can be somewhat expensive (calculating distances, averages over previous frames etc) It is best done in some process other than the rendering process and is commonly done as part of the culling process for objects once they have been determined as visible

Load management needs accurate time on when rendering finishes

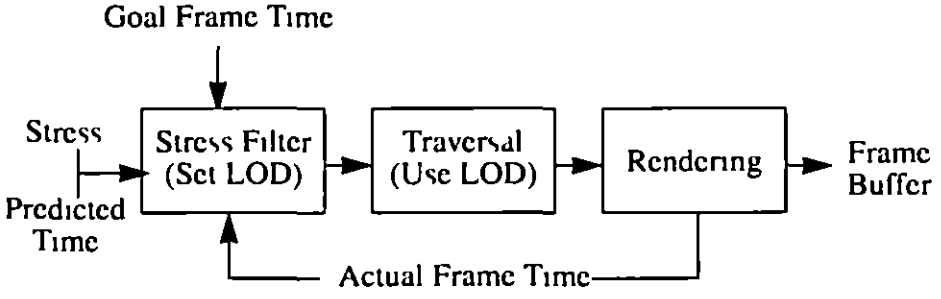


FIGURE 14 Load Management Closed Feedback Loop

LOD selection

Load management for graphics is an automatic way of saying "draw less One convenient way to do this is by applying a scaling factor to the *levels of detail* (LODs) of the database, using lower LODs when the system is over-loaded and higher LODs when it is under-utilized A simple mechanism for this is to simply scale the effective distance between object and eyepoint by a factor based on load when computing range-based LOD selections Analogous load-based scalings work for other criterion for LOD selection A hysteresis band can be used to avoid changing LODs when the load for the past few frames is staying within a bounded region or to wait until the load has strayed from the bounded region for some number of frames before changing to avoid quickly thrashing between high and low levels of detail This is the mechanism used by IRIS Performer[Rohlf94]

Use LOD mechanism that addresses real rendering bottleneck

This technique alone is quite effective at reducing load in the geometry subsystem because object levels of detail are usually modeled with fewer vertices and polygons The raster subsystem may see some load reduction if lower levels of detail use simpler fill algorithms, however, they will probably still require writing the same number of pixels and could even cover more pixels for shapes that are boxier than their more finely modeled counterparts LOD transition algorithms also play a big part in load management as expensive LOD transitions could be the cause of further system load forcing more LOD activity

In visual simulation applications it is common to have objects that are known to be very central to the current task, such as the stripes or lights on a runway for a flight trainer LOD classes with their own additional scale factors (or more complex criterion) can be used to enable LODs to be used for all objects while keeping high priority objects immune to all but extreme stress conditions Another common trick is to aggressively scale down LODs as a function of distance so that distant objects are not drawn A distant fog band makes this less noticeable LOD selection algorithms can consider factors in addition to the basic criteria modeled into the database, such as existing atmospheric conditions that might obscure visibility, and rendering load

Dynamic Resolution

The types of load reduction discussed so far were based entirely upon the utilization of pre-modeled levels of detail This has several serious limitations First it requires the database modeling which often has to be done by hand, or at least hand-tuned Second, the effect of automatic load-based LOD selection is often quite non-linear and it is very difficult to ensure minor or major changes in load as desired This is particularly true if the objects are fill limited and visual simulation applications are very com-

monly fill limited The different levels of detail may not reduce raster computation If the system supports it dynamic video resolution (DVR) is an ideal way to address fill limitation DVR, as supported on the Silicon Graphics InfiniteReality, allows a per-channel bilinear-filtered zoom as part of the video scan out operation with no performance penalty or added latency The zoom scale factor can be changed on a per-field basis This allows the application to select a rendering resolution for each frame and for each eyepoint A very minor change in effective resolution can greatly improve the rendering time and the effect is completely predictable If an application would like to reduce the rendering time from 16msecs to 14 msecs then it will draw 12 5% fewer pixels, or simply scale each of the current width and height of the viewport for the next frame by the sqrt(1-reductionFactor) which in this case would be 94% This technique is database independent is linear in required change for desired effect, and produces predictable effects on performance

Rendering timing measurement and prediction

The best load management scheme is only as good as its input information and this is the measurement and/or prediction on rendering time Measuring the time of a single frame of real-time rendering can be quite impossible without hardware support The problem is similar to that in benchmarking in that you want to measure exactly how long it took for the graphics pipeline to rendering specific geometry In benchmarking you have the option to draw enough geometry to draw for a long enough time so that the resolution of the available clock and overhead in getting it does not corrupt the desired measurement Additionally the graphics pipeline must be flushed after rendering (glFinish) before taking the time measurement because very expensive rendering commands can sit buffered in the pipeline while the application returns immediately to take the time However, this often requires drawing for a second or more to hide this overhead and for a single frame in a real-time application we only have a few msecs The InfiniteReality has the ability to time rendering of specified geometry from within the graphics pipeline and return these timings to the host asynchronously with virtually no performance impact these timings are used by IRIS Performer in the feedback loop to drive the load management system

8 3 Reducing Latency

Reducing frame rate helps to reduce latency, but not necessarily enough Frame rate can be reduced with increased pipelining to allow for more aggressive scene management and faster rendering However increased pipelining may increase the total latency Approaches to reducing latency

independent of pipeline depth focus on the eyepoint This technique relies upon the observation that it is the eyepoint latency that is so critical in avoiding simulator sickness Having objects update in the scene at longer latencies, while not ideal, is far less problematic

One class of algorithms to handle eyepoint latency makes use of the fact that often the eyepoint is already the result of some fairly intensive computation to smooth the input from potentially noisy input devices Some additional computation can be done to take into account current speed and direction and the rendering rate to forward extrapolate the eyepoint for a future point in time to use in culling and rendering This can effectively remove several pipeline stages of latency

Other methods are based upon allowing certain external inputs, namely viewer position, into later stages of the graphics pipeline If the maximum pan (or slew) rate is known a priori then the culling volume can be bloated to accommodate any potential viewpoint change that could arise during the culling stage of the pipeline and a new eyepoint position can be fed in for use in rendering after the culling stage Finally, again with a slightly enlarged viewing projection very minor changes in view can be accommodated even after rendering the image in the framebuffer Hardware support for this including image warping for perspective correction is suggested by [Regan94] with the address recalculation pipeline The viewer orientation is sampled after rendering by a separate pipeline that runs at video refresh rate to produce the output RGB stream for video Additionally only objects that are moving need to be redrawn as the viewer changes his orientation A simpler algorithm that simply adjusts the origin of the visible area of the framebuffer for video scan-out is suggested by [Riner92] The InfiniteReality dynamic resolution video hardware can support the panning algorithm with the additional ability to zoom the desired area to accommodate for forward or backward motion

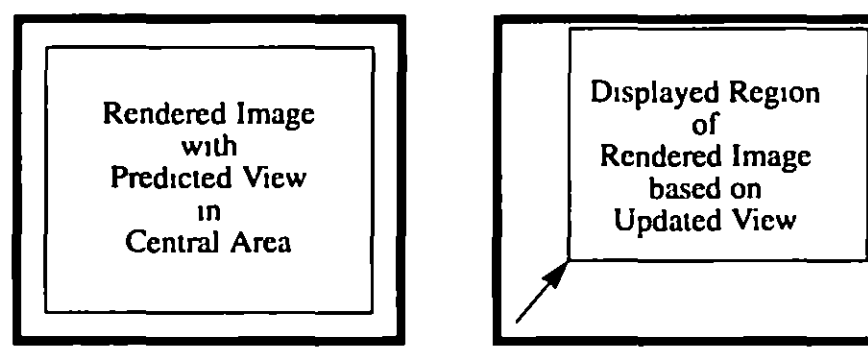


FIGURE 15 Dynamic Video Resolution with Pan and Zoom

9 *Visual Simulation Techniques for Database Tuning*

Tuning databases is as important (and often just as much work) as tuning the application The database hierarchy needs to be structured to optimize the major traversal tasks Information cached in the database hierarchy can reduce the number of dynamic operations that must be done by traversals Finally the modeling of the database geometry should be done with an understanding of the performance characteristics of the graphics pipeline

9 1 Spatial Hierarchy Balanced with Scene Complexity

The major real-time database traversals are the scene management traversals and collision traversals Both benefit by having a database that is spatially organized, or is coherent in world space These traversals eliminate parts of the scene graph based on bounding geometry If a database hierarchy is organized by grouping spatially near objects, then entire sub-trees can be easily eliminated by the bounding geometry of a root node If most nodes have bounding geometry that covers much of the database then an excessive amount of the database will have to be traversed

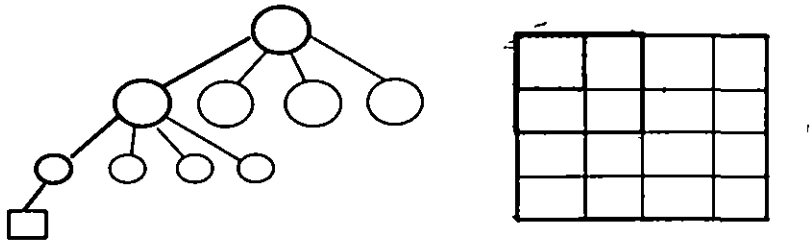


FIGURE 16 Scene graph with Spatial Hierarchy

It is additionally helpful to have a hierarchy based on square areas so that simple bounding geometry such as bounding spheres, can be used to optimize the traversal test

The amount of hierarchy put in the database should balance the traversal cost of nodes with the number of children under them A node with few children will be able to eliminate much of the database in one step However, a deep hierarchy might be expensive to maintain as objects change and information must be propagated up the tree

9 2 Database Instancing

Instancing in a database is where multiple parents reference a single instanced child which allows you to make performance/memory trade-offs

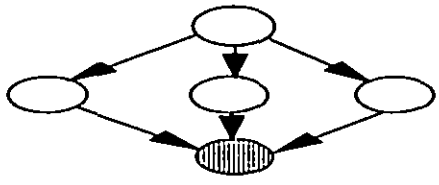


FIGURE 17 Instanced Node

Instancing saves memory but prevents a traversal from caching traversal information in the child and also prevents you from flattening inherited matrix transformations To avoid these problems, IRIS Performer provides a compromise of *cloning* where nodes are copied but actual geometry is shared

9 3 Balancing the Traversals

A spatial quad tree datastructure optimizes most graphics application trafersals

The amount of geometry stored under a leaf node will affect all of the traversals, but there is a performance trade-off between the spatial traversals and the drawing task Leaf nodes with small numbers of polygons will provide a much more accurate culling of objects to the viewing frustum, thus generating fewer objects that must be drawn This will make less work for the rendering task, however the culling process will have to do more work per polygon to evaluate bounding geometry If the collision traversal needs to compute intersections with actual geometry, then a similar trade-off exists fewer polygons under a leaf node means fewer expensive polygon intersections to compute Note that most database traversals (collision, intersection culling, LOD even audio) in a graphics applications use viewer eyepoint and object position and benefit from a spatial organization to the database

9 4 Modeling to the Graphics Pipeline

The modeling of the database will directly affect the rendering performance of the resulting application and so needs to match the performance characteristics of the graphics pipeline and make trade-offs with the database traversals Graphics pipelines that support connected primitives, such as triangle meshes, will benefit from having long meshes in the database However, the length of the meshes will affect the resulting database hierarchy Long strips through the database will not cull well with simple bound ing geometry

Objects can be modeled with an understanding of inherent bottlenecks in the graphics pipelines Pipelines that are severely fill limited will benefit from having objects modeled with cut polygons and more vertices and fewer overlapping parts which will decrease depth complexity

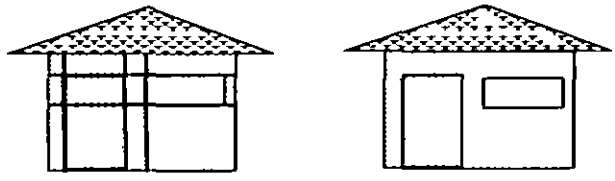


FIGURE 18 Modeling with cut polygons vs overlapping polygons

Pipelines that are easily geometry or host limited will benefit from modeling with fewer polygons

9 5 Use Effective Resolution Where you Need it

Calligraphic points can be used for depth buffered infinite resolution insets

It is a natrual tendancy to take the maxim resolution requirement and force the entire applicaton to runat that resolution However some parts or views of the simulation may not need the maximum level of resolution Some systems support different video output formats with different resolutions and even different timings outputing from the same framebuffer source Resolution requirements are also often nade for specific elements of the database such as firing targets These targets can potentially be rendered separately at higher resolution and combined using video matting techniques with the background scene Calligraphic points, such as is on Silicon Graphics InfiniteReality can offer an infinite-resolution solution that can also be automatically properly depth buffered against the main scene

10 Texture Mapping Techniques and Management

10 1 Texture Replaces Geometry

10 2 There are modeling tricks that can reduce database complexity Many focus on using textured polygons to simulate complex geometry It is especially useful if the graphics subsystem supports the use of *alpha textures* where a channel of the texture image indicates the transparency of the object Texture can be made as cut-outs for things like fences and trees The OpenGL AlphaFunc command can be used to reject texels/pixels of very low opacity and thus low contribution to even further improve the performance of these textures The color of the background of such

textures should be the average color of the image so that averaging across the image boundary for MIPmapping for minification does not produce a badly discolored silhouette. A sharpening filter for the alpha component for magnification of these textures prevents them from becoming blurry and transparent upon magnification with close inspection. The trees in Figure 19 use a bilinear magnification filter but the right hand trees also use sharpen on the alpha component.

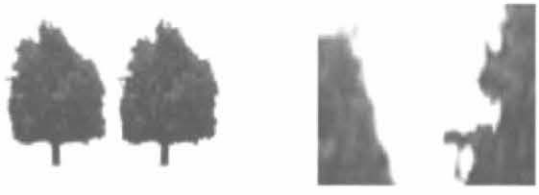


FIGURE 19. Texture with Sharpen MagnificationFilter

10.3 Use of the MIPmapping S and T biasing controls (also OpenGL features) enable one to prevent blurring under minification when the texture is viewed orthogonally to its expected use and is experiencing simultaneous extreme minification in one dimension and magnification in another, as can happen with roads if they are not modeled with a short and wide texture. Anisotropic texture filters are another more automatic way to handle this situation at the possible cost of increase in fill time based on the ratio of distortion as more texture samples are required for the proper averaging. In Figure 20, the roads are applied with the same repeating road texture viewed in sharp perspective such that texels are being heavily minified in T but still magnified in S. The road on the left is applying LOD biasing in the T dimension to correct for the repeat factor. The road on the right is the uncorrected road and the strips blur quickly because low-resolution texels were choosen based on the worst case minifiaction factor. The LOD biasing operation incurrs no performance or texture memory penalty.

FIGURE 20. MIPmap LOD Biasing for Anisotropic Correction

Billboards

Textured polygons as single polygon billboards are additionally useful *Billboards* are polygons that are fixed at a point and rotated about an axis or about a point so that the polygon always faces the viewer Billboards are useful for symmetric objects just as light posts and trees, and also for vol ume objects such as smoke Billboards can also be used for distant objects to save geometry However the managing of billboard transformations can be expensive and impact both of the cull and draw processes unless there is direct hardware support for them

Billboard textures can employ some additional tricks for efficiency such as palletization discussed in the next section

10 4 Texture Management

Understand the system capabilites for texture mapping paging bandwidth memory management and texture fill rate

Texture can quickly become the dominant element of scene complexity Texture mapping envolves the entire system rendering hardware, memory management, and host->graphics and disk->graphics paging bandwidth Some systems force textures for an entire database to fit in hardware tex ture memory which makes efficient use of hardware texture memory essen tial Changing between textures can be an expensive state operation for some graphics pipelines so it is important to understand the cost of this operation

One technique that can help with both efficient use of memory and texture changes is *palletization* With this technique multiple images are packed into a single texture and the texture coordinates of object polygons are used to select the proper image within the larger super texture This technique can only be used with non-repeating textures and requires that textures of like components and formats be used in the pallet This technique also works better for non-MIPmapped textures as with MIPmapping there can be bleeding between textures at lower levels of resolution from the filtering unless sufficient space is left between the images Palletization removes the cost of texture state change operations Additionally, palletization can remove sizing restrictions (such as power of 2) on individual images and can eliminate tiny textures both things that can cause waste of texture memory

Texture paging can allow hardware texture memory to appear almost infi nite To work in a real-time application, textures need to be able to be paged in pieces so that parts of textures can be trickled in over a number of frames as time allows The time for texture loads also needs to be known or predictable so that loads can be scheduled so as to not cause overrun Tex tures are then scheduled for loading in advance of their need based on vehi cle speed and database position Default textures can be left in texture

memory and used if the proper texture is not yet ready for use Textures that would potentially be needed with very low latency, such as explosions should also be left in texture memory These operations all require some ability to control the management of texture memory what textures are resident what textures are swapped out and what textures are loaded and at what rate

10 5 Multipass texture techniques

General texture support for loading and creating textures combined with pixel blending operations can be combined in multiple passes to combine effects These multiple passes can be implemented and hidden in hardware for specific common cases but if the system supports it general use of multipass texture mapping provides a powerful mechanism for special effects Additional passes (in hardware or software) will have additional expense and so time for their use must be allocated in the rendering budget

Multipass texturing is a powerful and general technique and with an easily assessed cost

Multipass algorithms can be applied to individual objects, such as using blending to add in a specular highlight or modulate in surface attribute shading to the already textured object such as the multipass texture bump-mapping approach suggested in [Peercy97] Multi-pass techniques can also be full-scene where a rendered scene is simply input to a texturing algorithm to produce a distorted version such as for environment mapping environmental effects such as fog and rain, distortion to match a display surface, or to simulate a special output device The key capabilities for supporting generalized multipass texturing is support for copying framebuffer memory to textures, and preferably the ability to update just parts of textures The OpenGL command in use would be `glCopyTexSubImage2D()` The additional capability of being able to add hardware such as the use multiple graphics pipelines in a parallel or pipelined fashion as discussed in "Multiprocessing of Entire Graphics Pipelines" on page 20 can enable the use of these general techniques without special hardware support

Environment Mapping

The use of environment maps as texture maps and geometric surface normals for indexing into the environment texture was originally introduced in [Greene86] and further developed in [Voor94] It has since become as standard technique with its own special OpenGL texture generation mode GL_SPHERE_MAP, where the normals are used to index into an environment that has been mapped onto a sphere Environment mapping is a generally useful technique to combine with other effects for producing realistic reflective surfaces The environment maps can produce texture effects on the object surface based on surface normal, or else can be correct renderings from the actual object environment for producing realistic (dynamic)

reflections. The spherical environment map can be created by taking the 6 different viewpoints from a given eyepoint that would produce an environment mapped cube and then distorting these images to fit into a single hemisphere. The distortion, pictured below, may appear to be challenging to compute.

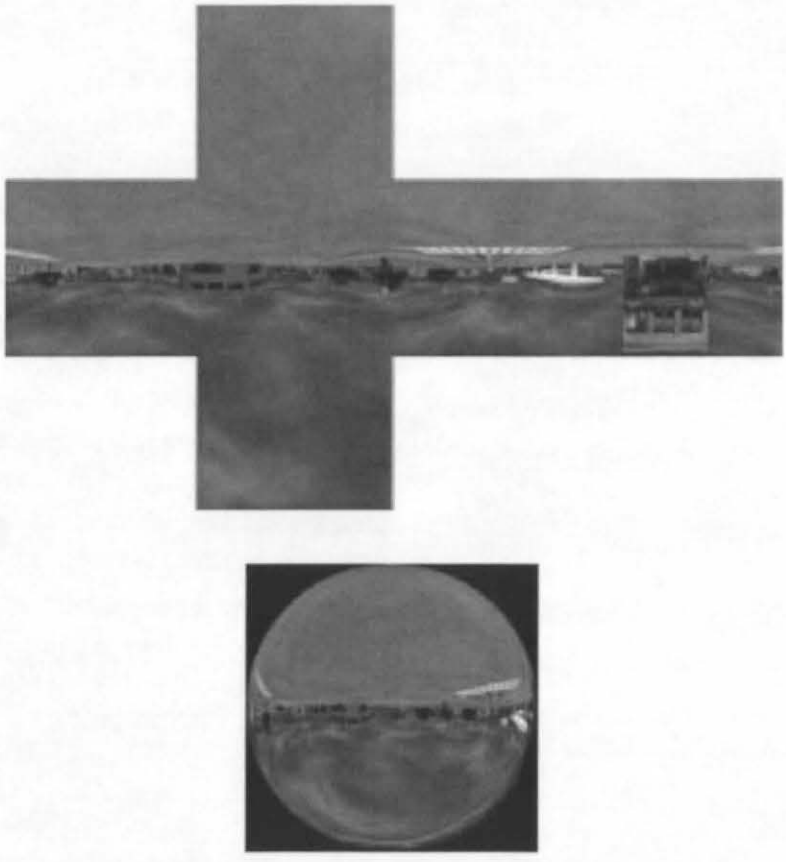


FIGURE 21. Computing a Spherical Reflection Texture Map

Texture mapping can be used for the intermediate computation to produce the distorted image for the final texture. A cube is created with texture coordinates of the faces matching the cube environment map. However, vertices are then added with interpolated texture coordinates to bloat this cube's faces out to the boundaries of the unit sphere. Finally, the vertices are transformed to pull them forward into the plane of one pole based on their angle with the viewing vector towards that pole and thus flattening the sphere. The mesh is then rendered to produce the distorted cube map image and this image is then used as the environment map texture. On systems with sufficient texture loading speed, these environment maps can be created in real-time.

The spherical distortion will sacrifice some parts of the environment map for the benefit of using others such as infinitely compressing the pane behind the eye to the edges of the hemisphere Potentially two environment maps in combination with backface removal can resolve these artifacts if they are significant to the application Future hardware may one day improve this technique

Textured Lighting Effects

Needed lighting effects often include specular highlights, cast spotlights and shadows These can be produced at high-quality with the use of general texture mapping techniques

Use projected textured spotlights for landing lights

Cast spotlights are often used for the landing lights of airplanes The pattern of a cast spotlight can be drawn into a texture (such as shaded white circle with black background that contributes no light) and applied to surface geometry with a projective texture matrix from the view of the light The scene is rendered for each light with hardware lighting for the light color and using the texture map as intensity to modulate the image, drawing only areas that are lit Successive renderings for each light are summed

A shadow texture for an object can be projected from the position of the light onto destination objects using the standard projection techniques However for applications where shadows are really critical, such as plane refueling, pre-computed shadows are not sufficient The InfiniteReality supports a simple extension that enables the computation of shadows in real-time Texture-based shadows uses a depth comparison technique to determine if a given pixel should be in shadow or not The normal scene is rendered In an off-screen area the scene is rendered from the viewpoint of the light and the resulting depth buffer is transferred to a texture depth map The scene is then rendered with this texture map being applied with a texture projection matrix to correct for the mapping from the light and texture coordinates that are generated to map the world into the [1, 1] cube The texture function is one that does a comparison between the interpolated and perspective-corrected generated value of the R texture coordinate representing depth Depth value of the texture and image values less than or equal to than those in the texture map are in the light and those greater than the texture map are in shadow Lit areas receive 1 0 for their texel color The InfiniteReality also supports a shadow ambient color to use other than the 0 0 that would normally result The shadow pass can then be modulated with a previous lit pass to dim areas that are in shadow

Hardware per-pixel fog using a black fog color can be applied in addition to range-attenuate lighting affects when light sources are near the eye

Texture based Range Correct Fog

Three-dimensional textures are useful for defining volumetric environments A reasonable approximation to range correct fog can be achieved with the definition of a three-dimensional texture volume where each layer of the texture is the map total fog intensity seen by the eye at a given elevation looking radially out (S coordinate) and for a given elevation (T coordinate) This map is actually stored in log space so that it is reasonable to store summed intensity values in the maps and interpolate between two maps for eyepoints at intermediate elevations A texture look-up table is then used to exponentiate the resulting density getting the proper multiplicative effect between fog layers This method then produces as an additional scene pass per-pixel range and angle correct fog blending values that can then be modulated with a fog color and added to the rendering of a scene already in the framebuffer The fog pass is blended with a depth function of Z-EQUAL so that only the fog values of visible surfaces are used This method uses the addition of a new texture coordinate generation function on the InfiniteReality a texture coordinate is based on radial distance from a defined eye-point relative to a defined up vector

10 6 Clipmapping — Beyond MIPmapping

The texturing techniques discussed thus far are addressing the affects and complexities produced by using individual textures for individual scene elements (trees, smoke sides of houses spotlights) However the most significant object in most visual simulation applications is the terrain and using individual textures for the terrain either produces a very generic and potentially cartnoonish or repeated or patterned look, or else it must use a tremendous number of textures with a very complicated texture management scheme

Managing a terrain with photorealistic imagery as individual textures is hugely complicated for both the application and the database to manage Traditionally the terrain is tiled into cells referencing separate individual textures that at the lowest level use MIPmapping for their minification filtering and at higher levels use again MIPmapped separate textures to form a hierarchical LOD structuring of the database

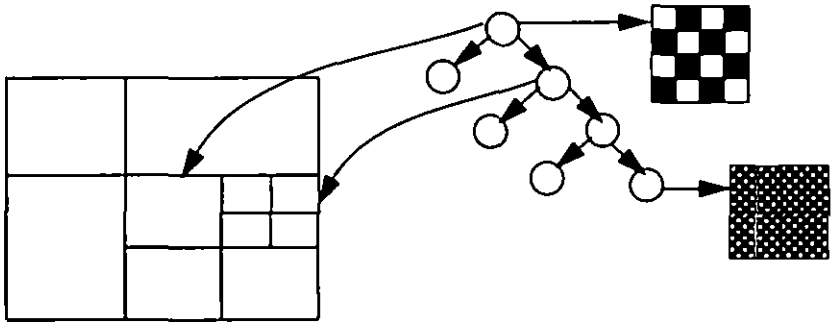


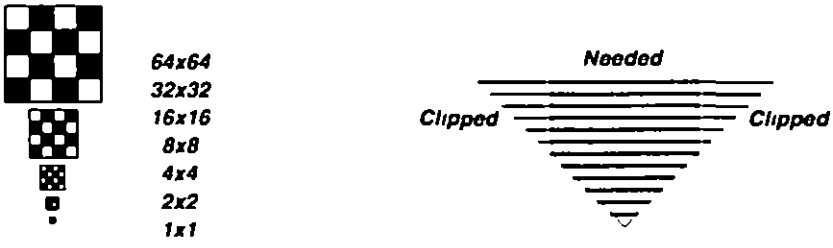
FIGURE 22. Terrain LOD Cells with textures of Matching Image Resolution

Decouple texture and geometry to enable smooth LOD on both

Multiple problems must be dealt with in this type of framework Each individual texture is a MIPmapped texture This produces high quality filtering within a texture as it is minified in a perspective view and neighboring texels from each of two surrounding levels of resolution are averaged to produce a textured pixel However at texture boundaries the filtering of one texture does not have access to the texels of the neighboring texture and so the filtering will not match between two neighboring textures in the terrain unless extreme special care is taken in both the generation of the textures and the mapping of terrain vertices into the textures This tightly couples the terrain modeling to the exact textures in use and any change to either terrain or textures can cause significant rework This tight coupling between terrain and the image data also makes it very difficult to implement high-quality terrain LOD mechanisms as the terrain at each LOD must fit exactly to the texture images of that LOD making continuous terrain LOD nearly impossible In addition, the texture memory management of the individual textures can be quite complex Each texture has its own pyramid of levels of resolution requiring with a naive approach many different sizes (some trivially small as 1x1) of allocated areas in texture memory which can easily lead to fragmentation If the individual textures are too large, there may be much wasted texture memory as many of the levels of a given texture may not be used

The Silicon Graphics InfiniteReality supports a first implementation of Clipmapping, a technique for moving beyond MIPmapping to supporting infinitely large textures that can use a small amount of hardware texture memory and decouple geometry from imagery A single texture can be millions of texels on a side allowing one texture to be used for an entire database This allows terrain vertices to reference anywhere in the terrain texture and thus decouples the terrain modeling from the texture

Clipmaps define a potentially visible clipped area within the full virtual MIPmap that would correspond to the original texture This clipped area need only be as large as the amount of high-resolution data that could possibly be visible at one time which will at worst be limited by the display resolution (such as 1K x 1K) but could be chosen to be much smaller A stack of some number of potentially visible levels in this clipped region is then kept in texture memory and this stack roams through the full virtual texture with the viewer s current position For a texture that is 8 million texels on a side, a full 1Kx1K clip stack from the 1x1 level up to the highest resolution level would use about 00006% of the full virtual texture The memory requirements can be made constant by only keeping a fixed number of levels in texture memory but the virtual texture can grow infinitely This technique and additional features are discussed in detail in [Tanner98]



Traditional full MIPmap pyramid Clipped stack of a cliptexture

FIGURE 23 Clipmap Stack vs MIPmap Pyramid

11 Tuning Tools

The proper tuning tools are necessary to simply evaluate an application's performance let alone tune it This section describes both diagnostics that you might build into your application, and some tools that the author has found to be useful in the past

11 1 Graphics Tuning Tools

A couple of standard tools for debugging and tuning graphics applications found to be useful on Silicon Graphics machines are `GLdebug` and `GLprof`, described in detail in [GLPTT92] and `OGLdebug` described in [OGLSGI97] `GLdebug` is a tracing tool that allows you to trace the graphics calls of an application This is quite useful because most performance bugs such as sending down redundant normals or drawing things twice have no obvious visual cue The tool can also generate C code that can be used (with some

massaging) to write a benchmark for the scene. `GLprof` is a graphics execution profiler that collects statistics for a scene and can also simulate the graphics pipeline and display pipeline bottlenecks (host, transform, geometry, scan-conversion, and fill) over the course of a frame. The `GLprof` statistics include counts for triangles in different modes, mode changes, matrix transformations, and also the number of polygons of different sizes in different fill modes.

11.2 System Tuning Tools

Some of the tools in the standard UNIX environment are also very useful. `prof`, a general profiler which does run-time sampling of program execution, allows you to find *hot spots* of execution.

Silicon Graphics provides some additional tools to help with system and real-time tuning. `pixie` is an extension to `prof` and does basic block counting and supports simulation of different target CPUs. `par` is a useful system tool that allows you to trace system and scheduling activity. Silicon Graphics machines also have a general system monitoring tool, `osview`, that allows you to externally monitor detailed system activity, including CPU load, CPU time spent in user code, interrupts, and the OS, virtual memory operations, graphics system operations, system calls, network activity, and more.

For more detailed performance monitoring of individual applications, Silicon Graphics provides a product called WorkShop that is part of the CASE-Vision™ tools which is a full environment for sophisticated multiprocess debugging or tuning[CASE94]. For monitoring of real-time performance of multiprocessed applications, there is the WindView™ for IRIX product based on the WindView™ product from WindRiver. WindView works with IRIX REACT to monitor use of synchronization primitives, context switching, waiting on system resources, and tracks user-defined events with timestamps. The results are displayed in a clear graphical form. Additionally, there is the Performance Co-Pilot™ product from Silicon Graphics that can be used for full-system real-time performance analysis and tuning.

11.3 Real-Time Diagnostics

Design tuning diagnostics into the application

The most valuable tools may be the ones you write yourself as it is terribly difficult for outside tools to non-invasively evaluate a real-time application. Real-time diagnostics built into the application are useful for debugging, tuning, and even load-management. There are four main types of statistics: system statistics, process timing statistics, statistics on traversal operations, and statistics on frame geometry.

System statistics include host processor load, graphics utilization, time spent in system code, virtual memory operations, etc. The operating system should allow you to enable monitoring and periodic querying of these types of statistics.

Process time-stamps are taken by the processes themselves at the start and end of important operations. It is tremendously useful to keep timestamps over several frames and then display the results as timing bars relative to frame boundaries. This allows one to monitor the timing behavior of different processes in real-time as the system runs. By examining the timing history, one can keep track of the average time each task takes for a frame, and can also detect if any task ever extends past a frame boundary. The standard deviation of task times will show the stability of the system. Process timing statistics from IRIS Performer are shown in Figure 24. Geometry statistics can keep track of the number of polygons in a frame, the ratio of polygons to leaf nodes in the database, frequency of mode changes, and average triangle mesh lengths. IRIS Performer displays a histogram of tmesh lengths, shown in the statistics in Figure 24.

FIGURE 24. Process and Database Statistics

Traversal and geometry statistics do not need to be real-time, and may actually slow traversal operations. Therefore, they should only be enabled selectively while tuning the traversals and database. Traversal statistics can keep track of the number of different types of nodes traversed, the number

of different types of operations performed, and perhaps statistics on their results. The culling traversal should keep track of the number of nodes traversed vs. the number that are trivially rejected as being completely outside the viewing frustum. A high number of trivial rejections means that the database is not spatially well organized because the traversal should not have to examine many of those nodes.

Additionally, IRIS Performer supports the display of depth complexity, where the scene is painted according to how many times pixels are touched.The painted framebuffer is then read back to the host for analysis of depth complexity. This display is comfortably interactive on a mid-range graphics hardware such as the IMPACT™ due to special hardware support for logical operations and stenciling. Thus, you can actually drive through your database and examine depth complexity in near real-time.

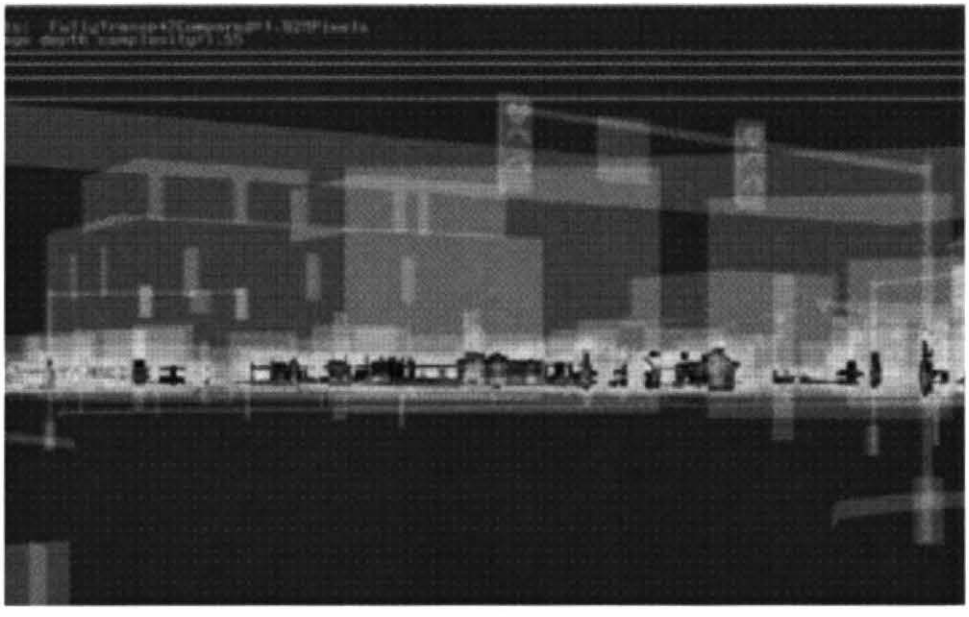


FIGURE 25. Pixel Depth Complexity Profile

12 Conclusion

Graphics workstations and low-end graphics platforms offer a wide array of options for balancing cost, performance, and image quality. Rich development environments plus real-time image-generator features like texture mapping, clipmapping, and full scene antialiasing make graphics workstations attractive platforms for both the development and deployment of visual simulation applications. High-end functionality is steadily making

its way down to low-cost solutions growing the potential applications of visual simulation solutions Understanding the target graphics architecture and designing for performance will enable you to get the scene quality and performance you paid for and the application you expect

13 Acknowledgments

The significant contributions from Jenny Zhao Angus Dorbie, and Michael Jones on visual simulation techniques and the extensive reviews from Allan Schaffer and Elizabeth Smith are all greatly appreciated

14 References

[Airey90] John M Airey John H Rohlf and Frederick P Brooks Jr "Towards image realism with interactive update rates in complex virtual building environments *ACM SIG GRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics* 24(2) 1990 pp 41 50

[Akeley89] Kurt Akeley The Silicon Graphics 4D/240GTX Superworkstation *CG&A* Vol 9 No 4 July 89 pp 71 83

[Akeley93] Kurt Akeley RealityEngine Graphics *Proceedings of SIGGRAPH 93* July 93 pp 109 116

[Cardul90] Frank M Cardullo and Yorke J Brown Visual System Lags The Problem The Cause The Cure *Proceedings of the 1990 IMAGE V Conference* June 1990 pp 31 42

[CASE94] *CASEVision™/WorkShop User s Guide* Document #007 1523 040 Silicon Graph ics 1994

[CN93] C Cruz Neira D J Sandin and T a A DeFanti Surround screen projection based virtual reality The design and implementation of the cave *Proceedings of SIG GRAPH 93* July 93 pp 135 142

[Deering93] Michael F Deering Leo A System for Cost Effective 3D Shaded Graphics *Pro ceedings of SIGGRAPH 93* July 93 pp 101 108

[Floriani95] L D Floriani and E Puppo Hierarchical Triangulation for Multiresolution Surface Description *ACM Transactions on Graphics* 14(4) 363-411 October 1995

[Foley90] James D Foley Andries van Dam Steven K Feiner and John F Hughes *Computer Graphics Principles and Practice 2nd Ed* Addison Wesley 1990

[Fuchs89] H Fuchs J Poulton J Eyles T Greer J Goldfeather D Ellsworth S Molnar G Turk B Tebbs L Israel Pixel Planes 5 A Heterogeneous Multiprocessor Graph ics System Using Processor Enhanced Memories *Proceedings of SIGGRAPH 89 July 89* pp 79 88

[Funk93] Tom Funkhouser and Carlo Sequin Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments *Proceedings of SIGGRAPH 93* July 93 pp 247 254

[GLPTT92] *Graphics Library Programming Tools and Techniques* Document #007 1489 010 Silicon Graphics 1992

[Greene86] Ned Greene Environment Mapping and Other Applications of World Projections *IEEE Computer Graphics and Applications* Vol 6 No 11 November 1986 pp 21 30

[Haeberli90] Paul Haeberli Kurt Akeley "The Accumulation Buffer Hardware Support for High Quality Rendering *Proceesings of SIGGRAPH 90 Aug 90* pp 309 318

[Harrell93] Chandlee B Harrell and Farhad Fouladi "Graphics Rendering Architecture for a High Performance Desktop Workstation *Proceedings of SIGGRAPH 93* July 93 pp 93 99

[Hoppe96] Hughes Hoppe Progressive Meshes *Processings of SIGGRAPH* 96 Jul y 96 pp 99 108

[Hoppe97] Hughes Hoppe View Dependent Refinement of Progressive Meshes *Processings of SIGGRAPH* 97 Augues 97 pp 189 198

[Lind96] Lindstrom Koller Ribarsky Hodges Faust and Turner Real Time Continuous Level of Detail Rendering of Height Fields *Processings of SIGGRAPH* 96 July 96 pp 109 118

[Luebke95] David Luebke Chris Georges Portals and Mirrors Simple Fast Evaluation of Potentially Visible Sets Proceedings of the 1995 Symposium on Interactive 3 D Graphics (April 1995 ACM Press

[Molnar92] Steven Molnar John Eyles and John Poulton PixelFlow High Speed Rendering Using Image Composition *Proceedings of SIGGRAPH 92 July 92* pp 231 240

[Montrym97] John Montrym Daniel R Baum David L Dignam Christopher J Migdal Infinite Reality A Real Time Graphics System *Proceedings of SIGGRAPH 97 August 97* pp 293 301

[OpenGL93] Jackie Neider Tom Davis Mason Woo *OpenGL Programming Guide* Addison Wesley 1993

[OGLSG197] *OpenGl on Silicon Graphics Systems* Silicon Graphics 1997

[PPG97] *IRIS Performer Programming Guide* Document #007 1680-020 IRIS Performer Silicon Graphics 1997

[Peercy97] Mark Peercy John Airey Brian Cabral Efficient Bump Mapping Hardware *Proceedings of SIGGRAPH 97 August 97* pp 303 306

[Potmesil89] Michael Potmesil and Eric M Hoffert The Pixel Machine A Parallel Image Computer Proceedings of SIGGRAPH 89 July 89 pp 69 70

[Rohlf94] John Rohlf and James Helman IRIS Performer A High Performance Multiprocessing Toolkit for Real Time 3D Graphics Proceedings of SIGGRAPH 94 July 94 pp 381 394

[Regan94] Mathew Regan Ronald Pose Priority Rendering with a Virtual Reality Address Recalculation Pipeline Proceedings of SIGGRAPH 94 July 94 pp 155 162

[Riner92] Bruce Riner and Blair Browder Design Guidelines for a Carrier Based Training System *Proceedings of the 1992 IMAGE VI Conference* July 1992 pp 65 73

[Tanner98] Christopher C Tanner Christopher J Migdal and Michael Jones The Clipmap A Virtual MIPmap *Processings of SIGGRAPH 98 July 98*

[Teller91] Seth Teller and Carlo Séquin Visibility Preprocessing For Interactive Walk throughs *Computer Graphics* Vol 25 Number 4 July 1991 pp 61 69

[Voorhie89] Doug Voorhies Reduced complexity Graphics *CG&A* Vol 9 No 4 July 89 pp 63 70

[Voor94] Douglas Voorhies and Jim Foran Reflection Vector Shading Hardware *Proceedings of SIGGRAPH 94* July 1994 pp 163 166

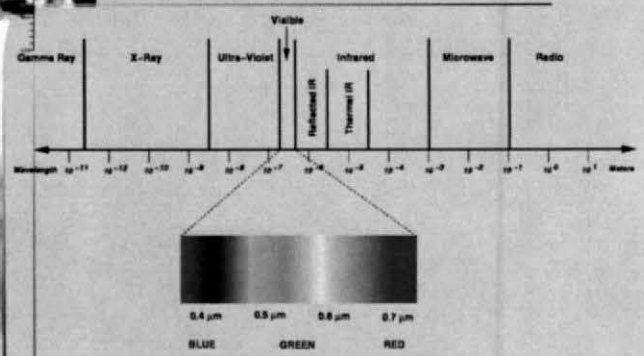
[Zhang97] Hansong Zhang Dinesh Manocha Tom Hudson and Kenneth E Hoff III Visibility Culling using Hierarchical Occlusion Maps *Processings of SIGGRAPH 97* August 97 pp 77 87

Requirements & Importance of Multispectral Applications

Graham Beasley

Real–Time Graphics for Visual Simulation: Advanced Techniques from the Top Down

The Electromagnetic Spectrum



Sensor Simulation
Infrared

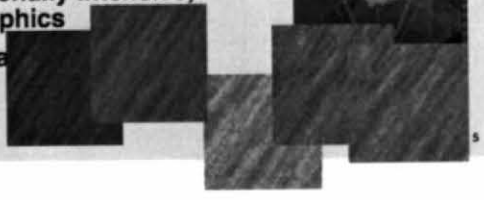
- Based on emitted thermal radiation (heat)
- Passive device
- Dynamic changes based on time of day
- Video effects
- Hot spots on models
- Typically low resolution
- Commercial products & toolkits available

Sensor Simulation
Night Vision Goggles

- Ambient light intensifier
- Passive device
- Blooming effects
- Video effects
- Typically low resolution
- Often use real goggles in a simulator – dark database
- Commercial products & toolkits available

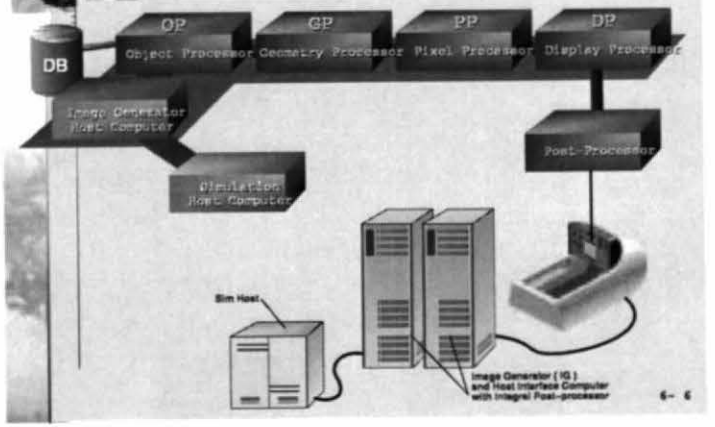
Sensor Simulation
Radar

- Transmitter sends out a signal, receiver picks up result
- Active device
- Beam sharpening modes
- Scan rates
- Computationally intensive, simple graphics
- Commercia available



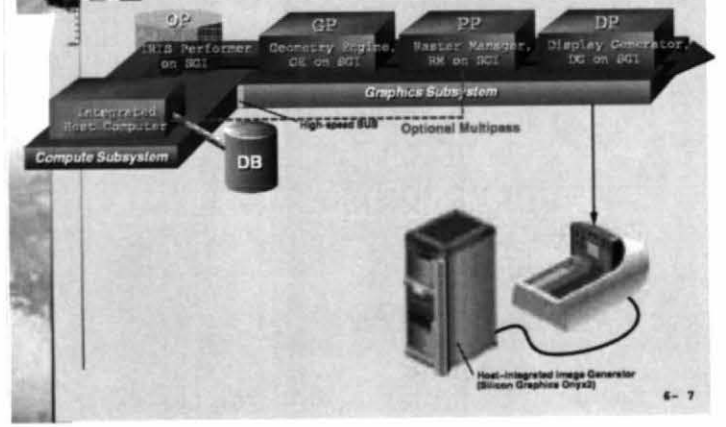
5

Traditional Image Generation



6- 6

Host-Integrated Computer Image Generation (HI-CIG)



6- 7

Quantitative IR Rendering in the SGI Graphics Pipeline

Seeing Accurately in the Dark

Stephen H Gersuk
Paradigm Simulation Inc
shg@ParadigmSim com

Infrared is ...

wavelengths beyond visible (>1μ)

commonly used in sensors because

- thermal emission supplants illumination as light source
- penetrates many obscurants (clouds smoke)
- detects thermal signatures of cultural objects & vehicles (stacks plumes hot chassis)

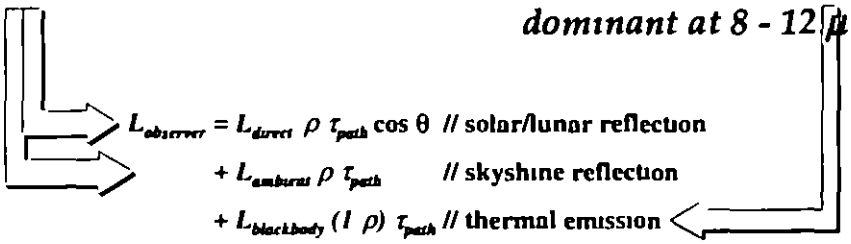
typically imaged at 3 - 5μ and/or 8 - 12μ because

- detectors exist (InSb and HgCdTe)
- twin bands enable discrimination between operating targets and burning targets

IR versus Visible Light

Most light at visible wavelengths arrives via reflection

At IR wavelengths, thermal emission is significant -- and dominant at 8 - 12 μ

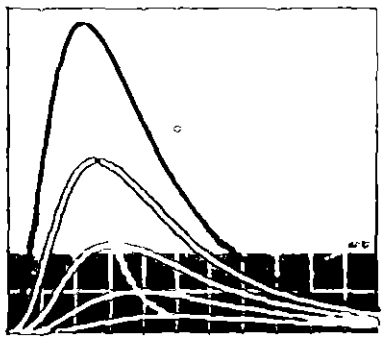


$$L_{observer} = L_{direct}\ \rho\ \tau_{path} \cos\theta \quad \text{// solar/lunar reflection}$$
$$+ L_{ambient}\ \rho\ \tau_{path} \quad \text{// skyshine reflection}$$
$$+ L_{blackbody}\ (1\ \rho)\ \tau_{path} \quad \text{// thermal emission}$$
$$+ L_{path} \quad \text{// path emission \& scattering}$$

Color and Brightness

Blackbody 'color' shifts with temperature

Total emittance varies with T^4



And as a Result ...

For visible light imagery

- Color & relative brightness are vary little over a wide range of environments and orientations
- Can empirically tune standard lighting functions

While for IR imagery

- Object appearance varies with TOD, weather azimuth slope
- Scene contrast may completely reverse from day to night
- Requires physics-based rendering for accurate results

Four-Part Solution

1 Reflectance textures

- Imagery classified into material map by texel color
- Texture computed as in band reflectance (0 1) for each texel for the underlying material

2 Polygon vertices assigned a material for thermal calculation

3 MOSART

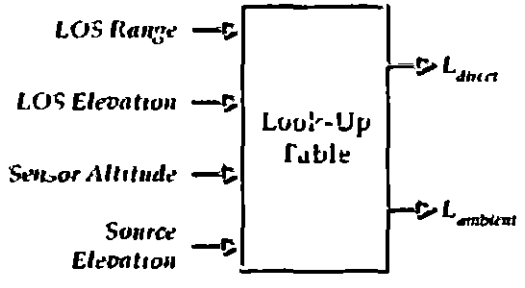
- Developed & maintained by USAF Phillips Lab
- Runs off line to create multivariate databases
- Databases interpolated on line for illumination fog, and vertex color

4 Reflected and emitted radiance summed via accumulation buffer in separate passes

Solar/Lunar & Skyshine Reflections (1)

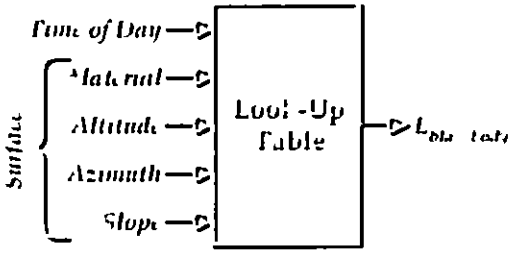
Solar/Lunar & Skyshine Reflections (2)

Look-up table indexed by view conditions interpolated for direct and ambient illumination per frame



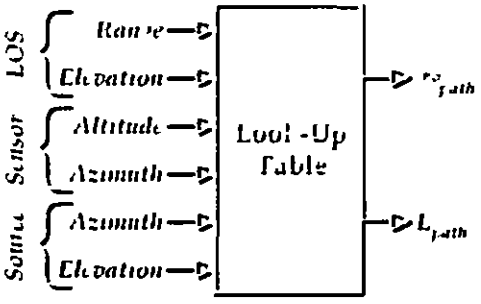
Thermal Emissions

Blackbody radiance indexed by TOD and surface properties sets vertex color of polygons



Path Emissions and Scattering

Path transmission and path radiance, indexed by everything, set fog color and density per frame



Conclusion

- *SGI graphics pipeline can be used to produce IR imagery*
- *Other codes can be used off-line to create multivariate databases interpolated on line*
- *Results are quantitive and subject to validation*
- *Acknowledgements*
  - SensorVision was developed by David Anding and Alex Szabo

Database Designs

Application Domains
Database Construction Process

Database Designs: Application Domains

Application Domains
Commercial Aircraft

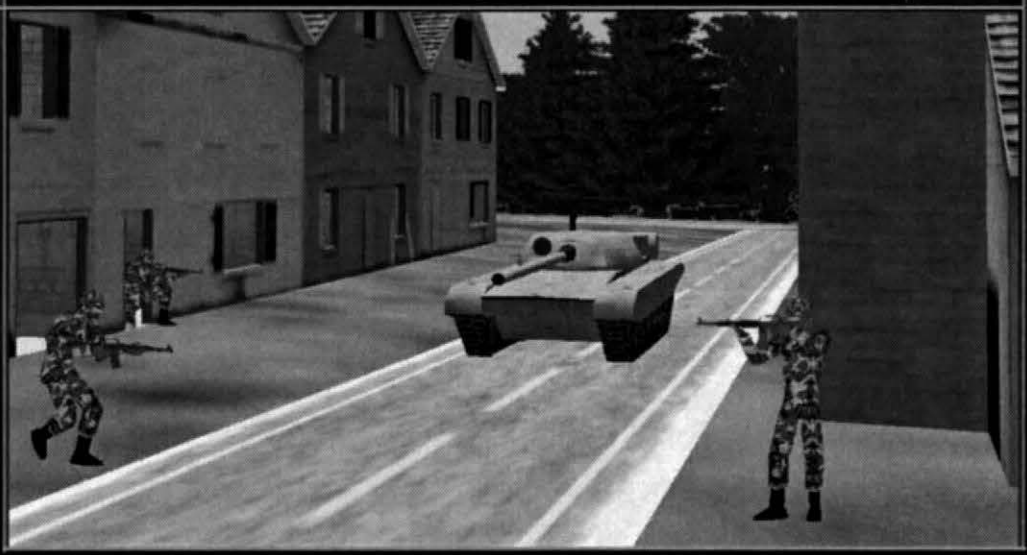
Application Domains
Military Aircraft

Application Domains
Rotary Wing

Application Domains
Driver Trainer

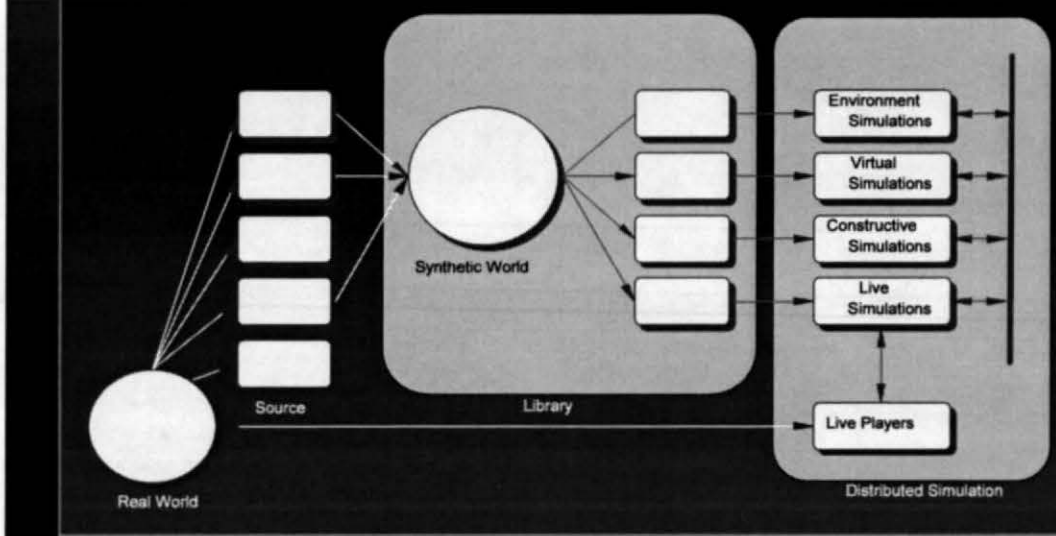
Application Domains
Combat/Maneuver Trainer

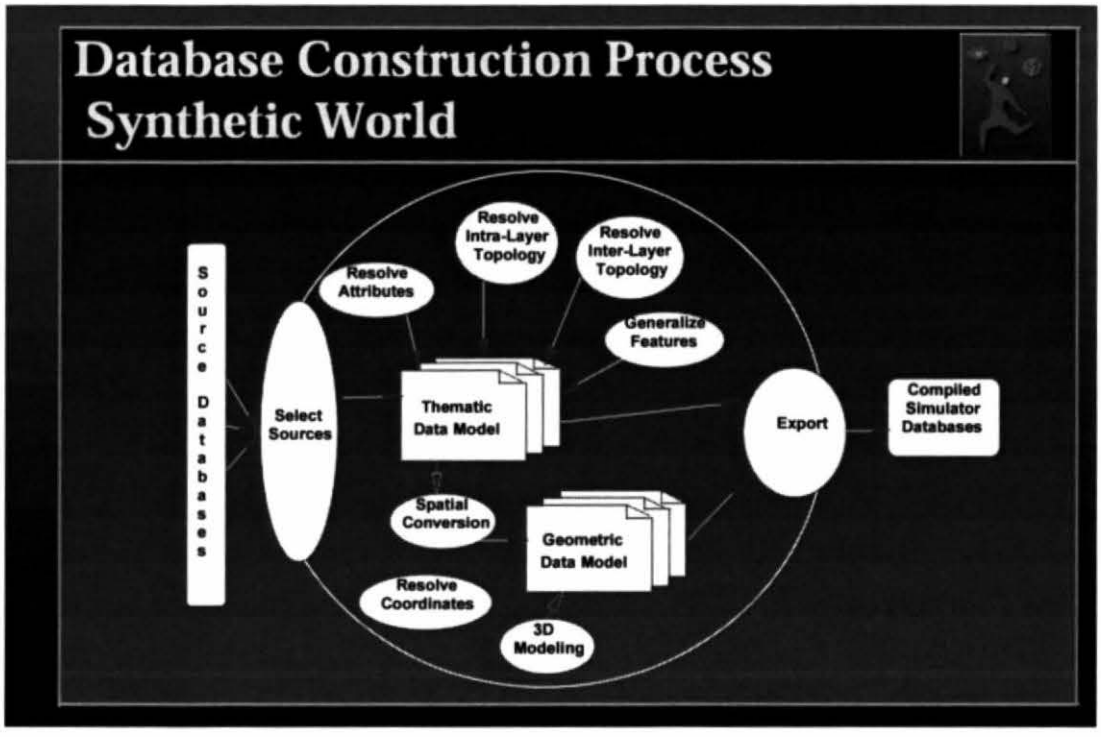
Application Domains
Dismounted Infantry



Database Construction Process

Database Construction Process World Reference Model




Database Construction Process
Synthetic World
Source Databases
Select Sources
Resolve Attributes
Resolve Intra-Layer Topology
Resolve Inter-Layer Topology
Generalize Features
Thematic Data Model
Spatial Conversion
Geometric Data Model
Resolve Coordinates
3D Modeling
Export
Compiled Simulator Databases

Large Area Texture & Terrain

"Above it all"

Requirements

Altitudes

- 3,000 to 30,000 feet and higher

Sight Distance

- 45 nautical miles or more

Fields of View

- 360 domes or 4 degree sensors

Coverage

- Hundreds or thousands of miles on a side
- Highly detailed target areas

Requirements

Frame Rates

- 60 Hz Visual and Thermal
- 1 Hz Radar

Correlation

- Visual
- Infrared
- Radar
- Constructive and virtual simulations

Database Components

Visual

- Image draped on terrain contour
- 3D models

Infrared

- Reflectance and/or radiance maps draped on terrain contour
- Thermal significant polygonal features w/ material codes
- Atmospheric model data

Database Components

Radar

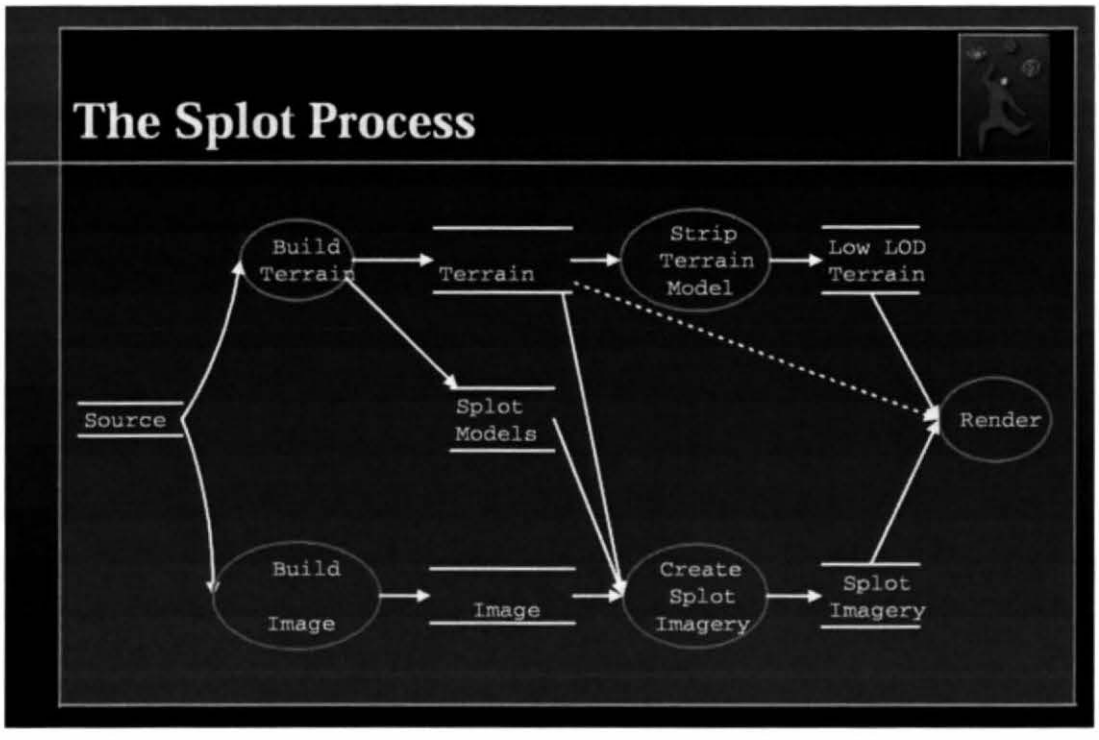
- Material maps draped on terrain contour
- Radar-significant polygonal features w/ material codes
- Radar cross section (RCS) lookup tables for models

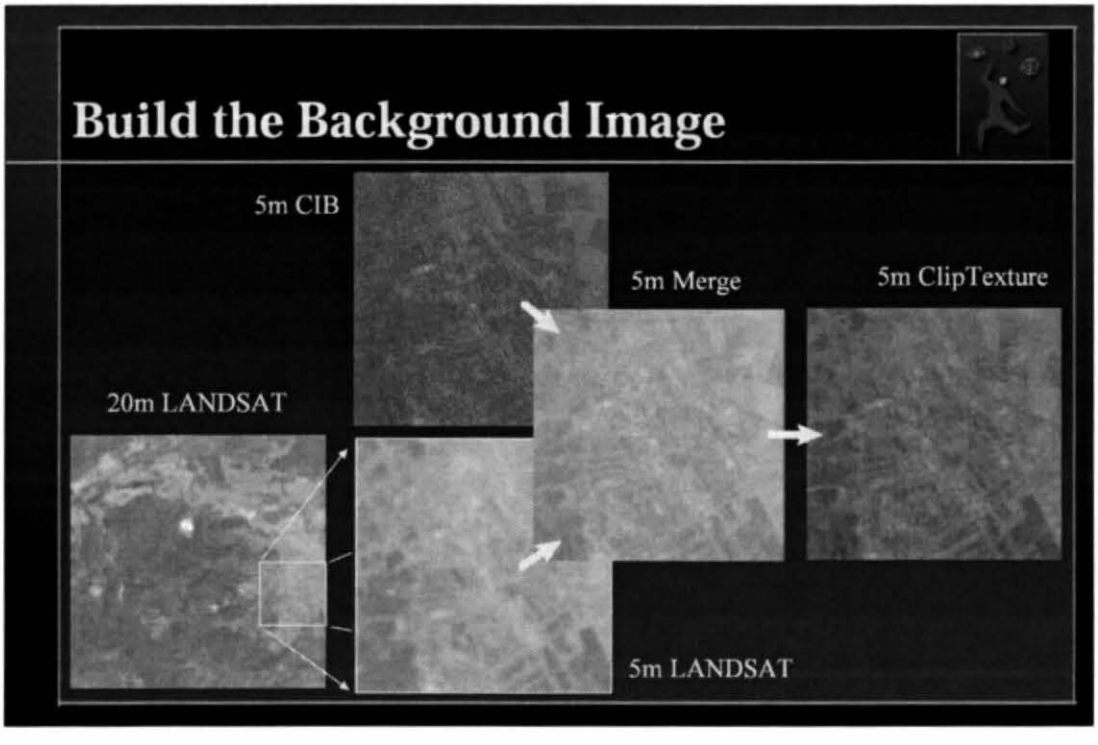
Constructive or Virtual Simulations

- Terrain Contour
- Vector Features

Splot

A technique for building a geospecific database with higher resolution and fewer polygons


The Splot Process
Build Terrain
Terrain
Strip Terrain Model
Low LOD Terrain
Source
Splot Models
Render
Build Image
Image
Create Splot Imagery
Splot Imagery


Build the Background Image
5m CIB
5m Merge
5m ClipTexture
20m LANDSAT
5m LANDSAT

Build Terrain

Terrain Skin

LOD Transition

2D Features

3D Features

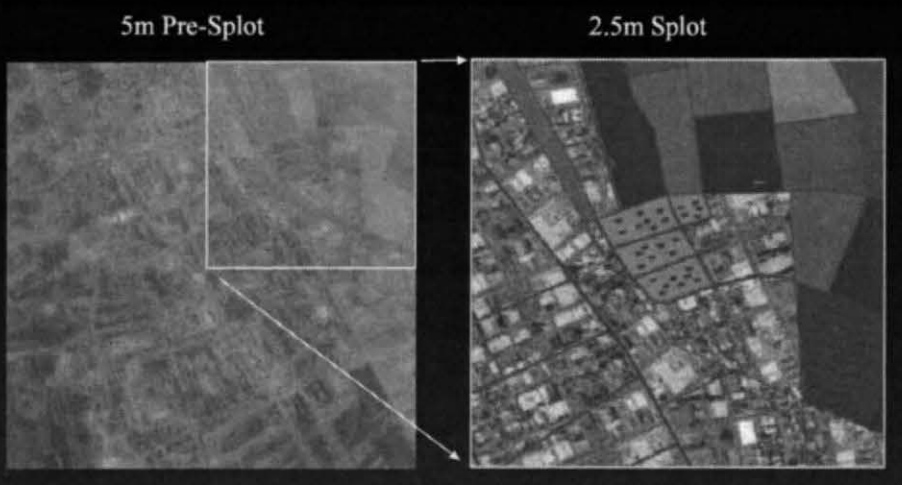
Prepare Terrain for OrthoCamera

Delayer

Splot Models

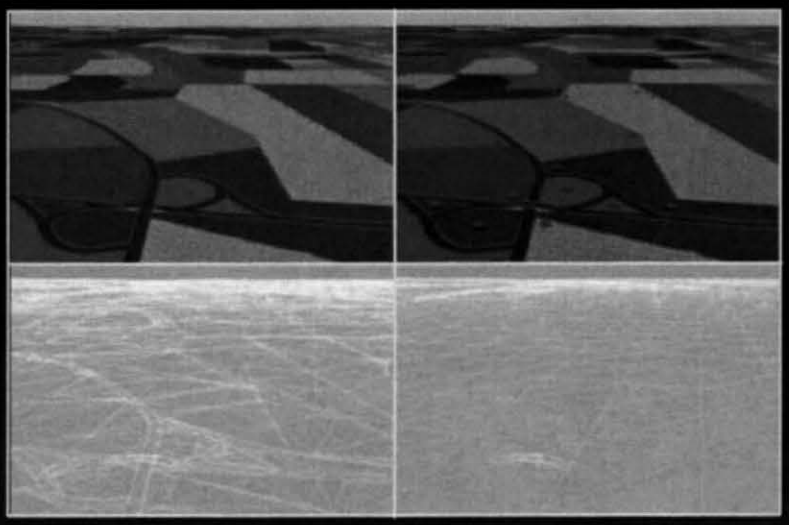
Not Splot Models

Create the Splot Imagery

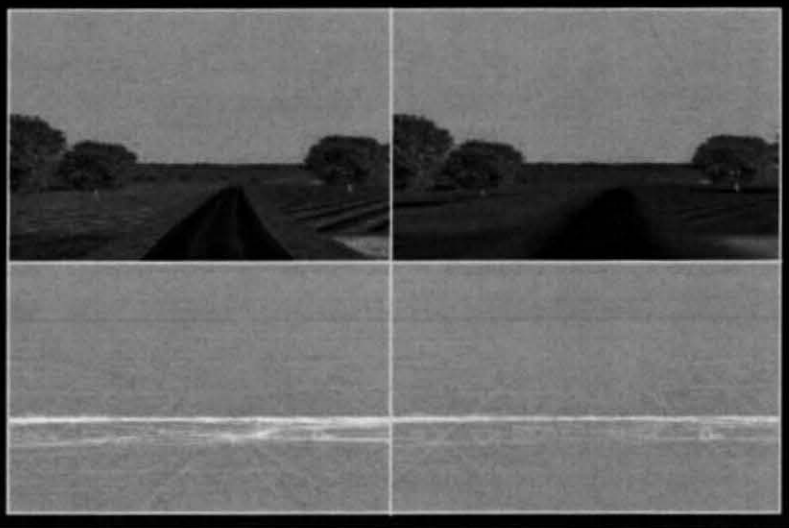


Render in Real Time

Pixels vs. Polygons



Content vs. Resolution



Infrared Clip Textures

Create the Reflectance Map

Use OrthoCamera to create Radiance Map

Build the Radiance Clip Texture

Render in SensorVision

Splot

"Synthetic photo-like ortho texture"

Dan E Brockway
Paradigm Simulation , Inc

Real-Time Graphics for Visual Simulation[1]
SIGGRAPH '98 Course

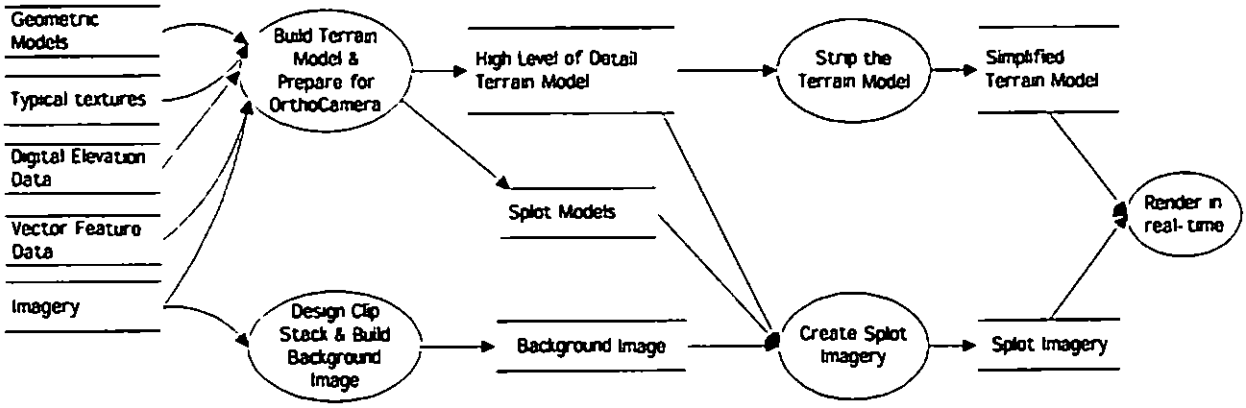
1 Introduction

Splot™ is a method for building a geospecific textured database with higher resolution than the source imagery and higher performance than the source models Splot renders and captures an orthographic projection of an enhanced scene and reuses it as textures
Scene resolution is enhanced by substituting higher resolution geotypical textures and models in place of lower resolution source imagery
Performance is improved by removing polygons that represent 2D cultural features on the terrain model and replacing them with their rendered images Splot provides essentially the same scene content with an order of magnitude reduction in the number of polygons
The Splot technique works equally well for database where there is no source imagery In this case a typical texture terrain model is built and imaged to create geospecific imagery

[1] Some of the techniques and methods disclosed here are covered by patents held by Paradigm Simulation inc Including U S Patent pending No 09/052 565

2 The Process

Essentially Splot is a process of building a terrain model and a background image (if imagery is available) and rendering that terrain into an orthographic channel Then taking the images and reapplying them as geospecific texture to a reduced polygon version of the same terrain model The following sections will cover the steps in detail



3 Design the Clip Stack

SGI Performer's clip texture mechanism manages paging large (many GB) textures from disk to system memory, then from system memory into texture memory To take advantage of this capability the geospecific texture must be built according to the following rules

1 The virtual size of the clip texture must be a square

2 The clip texture must have a power of two number of pixels on each side

3 Each level of the clip stack (called "image caches" by some) must have half number of pixels, per edge, as the more dense level "above it"

When designing a clip stack, identify your imagery resolution requirement to set the ground sample distance (GSD) a k a geographic size per pixel of each of the levels in the clip stack Remember rule 3 above This will affect many things First, all source imagery will need to be sampled to fit onto one level or another So, choosing a resolution set which aligns well with the source will save undue scaling (sampling) to align the source with the image stack Second, each more dense level of the clip stack requires four times as much disk space So, over specifying the resolution will cost in disk resources

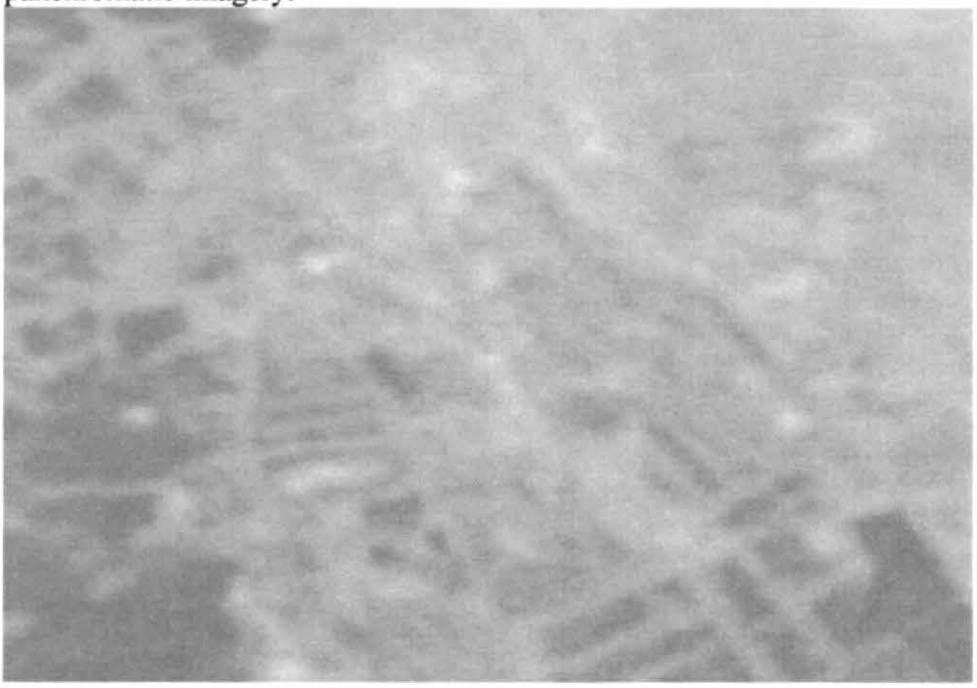
This section will use an example to illustrate the whole process The example clip stack has 19 levels The levels are referred to by ground sample distance if the pixels in the level The most dense level has 256k (262144) pixels on edge where each pixel covers is 2 5m of ground is called the 2 5m level

Merging Color and Panchromatic

In our example, we start with 5m panchromatic imagery. Covering a target area.

Separately, color imagery is derived from 25m LANDSAT and sampled to the 20m level of the clip stack.

The 20m LANDSAT is then sampled to 5m to register with the 5m panchromatic imagery.



...and merged with the panchromatic imagery to provide detail in the target areas..

5 Build the Terrain Model

Create a source terrain model (a k a database) Any terrain construction process can be used, but bear in mind that the performance advantage will only be realized if the 2D features can be stripped away to make a considerably higher performing database

Terrain Skin

Grid topologies mesh much better than Delaunay triangulations Adaptive grids where horizontal point selection is a function of elevation variations work very well

LOD

Splot is a very nice technique for masking level of detail (LOD) changes The highest LOD can contain all the polygonal geometry and the second LOD can be just an image of the geometry When you are far away, you can see the object as a subtle contribution to the imagery Then as you fly nearer the object becomes 3D

2D Features

Because 2D culture is stripped out, excessive numbers of polygons can be used to model 2D surface features Texturing of 2D features can also far exceed the usual run-time texture budget

3D Features

The real load management comes in 3D features Buildings, trees, power poles, etc can be managed with level of detail Avoid the use of large monolythic canopies They uses a considerable number of polygons and are dificult to manage under load stress conditions

Feature Codes

Attribute coding the polygons in the terrain model is highly recommended This enables software tools to strip polygons based on feature classes

6 Prepare the Terrain Model for OrthoCamera

Splot is an off-line process As such, the time to render the terrain model is not critical This gives rise to complex algorithms for generating shadows and lighting effects

Delayer

Resolving the priority of co-planar polygons is a difficult task for the real-time application Since Splot renders the terrain model in an orthographic channel A separate version of the database can be made which explicitly resolves the layering of co-planar polygons Again feature codes are useful here, to offset features which should render on top

Splot Models

Since the terrain model is used twice once to render the Splot Imagery and again (in lower detail) in real-time, 3D feature and object models can be different for each use

For example, in the orthographic view, a tree can be an alpha textured shadow or the image of fallen leaves. Buildings can be surrounded by gardens, fences, yard toys, whatever. The real-time model of a tree could then be a single billboard polygon, and the house a simple cube.
Any detail you add to the Splot models will become 2D. So take care to achieve the desired effect.

Remove Models Bridges, etc.

Some models should not exist when OrthoCamera is running. Imagine the orthographic projection of a bridge... Road stripes running doen the embankment and across the top of the water. Unless a bridge has an alternate representation, it should be mapped to the null model.

7 Create the Splot Imagery

Run OrthoCamera

Now that the terrain model has been prepared, systematically render the terrain into an orthographic channel and save the frame buffer to texture tiles to make a higher level of detail in the clip stack.
Since the terrain model can have arbitrary complexity, it can be imaged at any resolution desired. For example, the geospecific imagery background may have been 5m resolution, but the typical texture modeling could have centimeter resolution. Remember that each additional level costs 4 times the disk space of the current level.
The image below is a 2.5m resolution image generated from the 5m merge shown in section 4 above.

Rebuild the Clip Stack

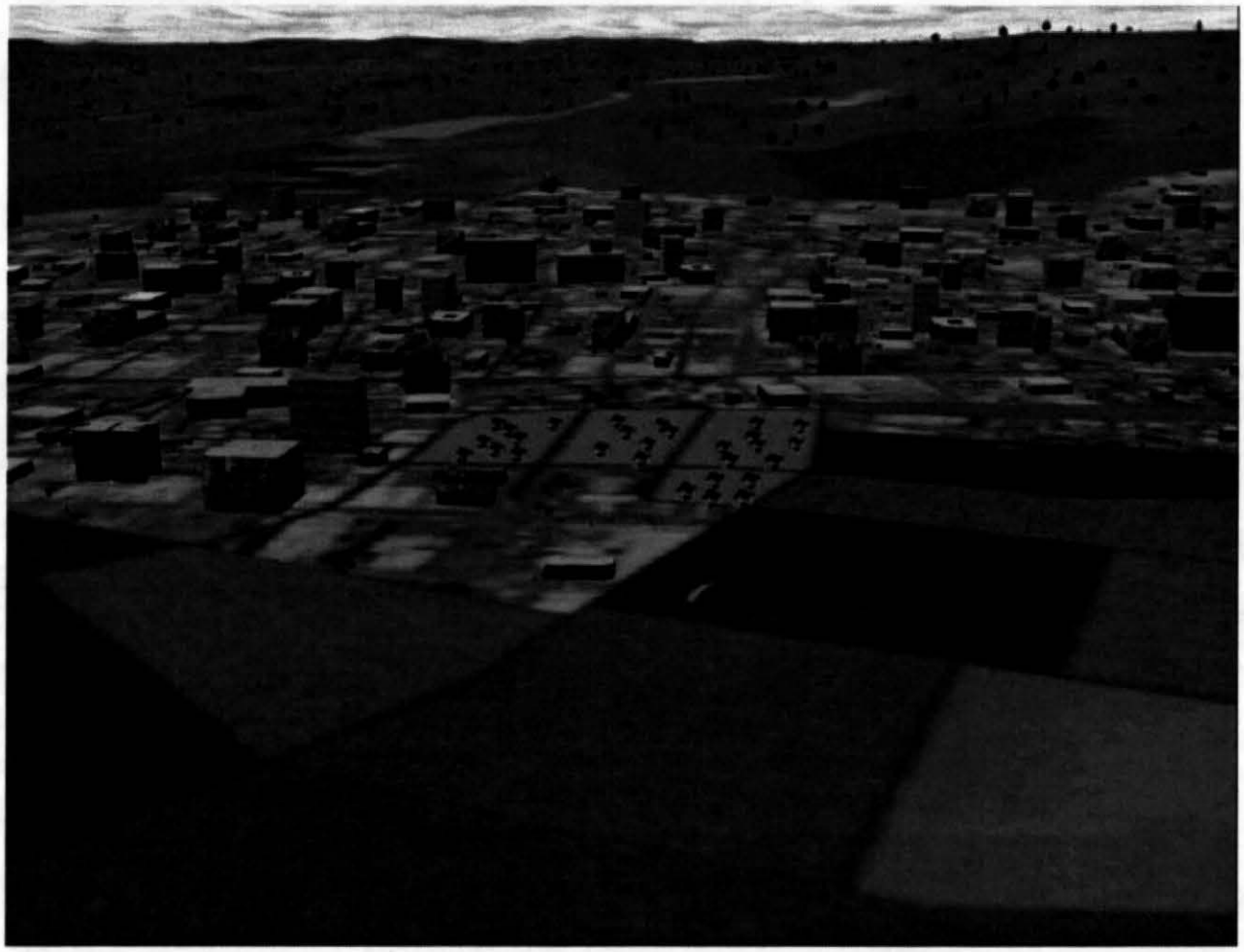
You may create Splot imagery for high detailed target areas or for the whole databases. In either case, you will now have a higher resolution image to integrate into the clip stack.

8 Strip the Terrain Model

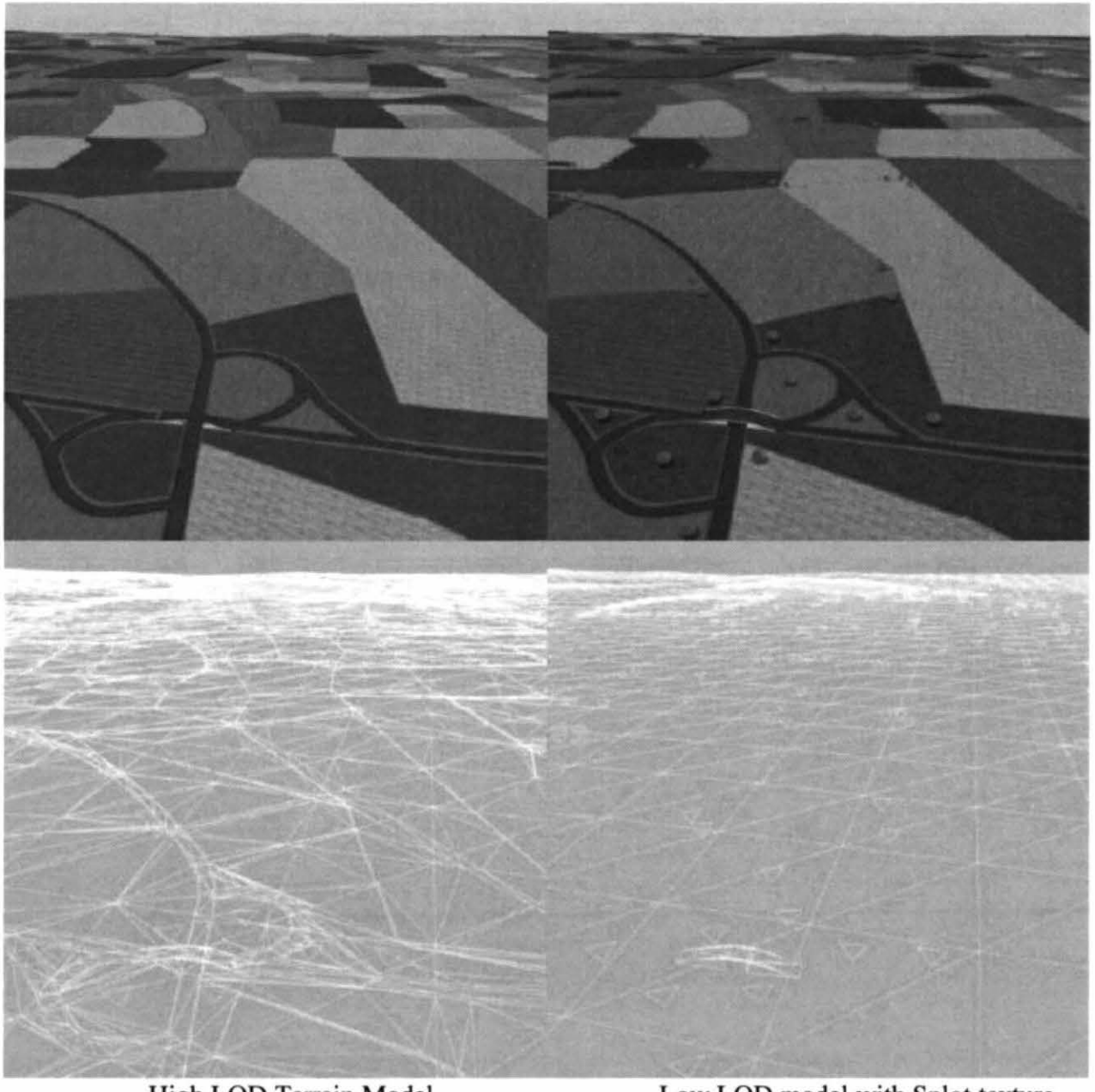
Now remove all polygons from the terrain model which are no longer significant contributors to the scene.
This can be done by removing all sub-face terrain polygons, all co-planar polygons, and all polygons having a specified set of feature identification codes.
See the images on the following pages.

9 Render in Real-Time

The following is a scene from the database built from the Splot imagery used in the examples above. The polygon budget is spent on 3D objects and not on the 2D features which are captured in the Splot imagery.



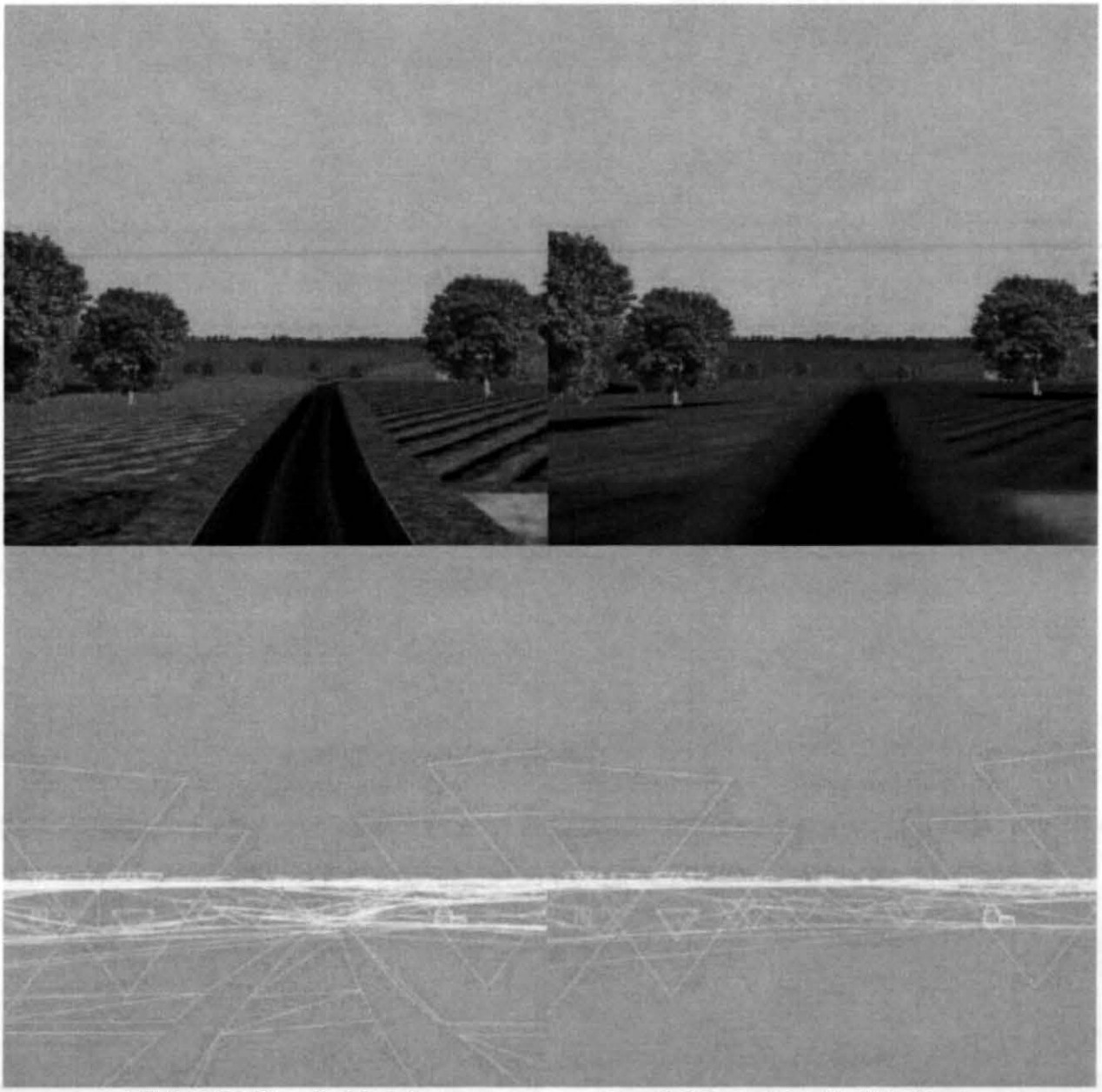
The following scenes show a database created without any background imagery, entirely typical texture. Notice the reduction in polygons with little sacrifice in apparent texture resolution.



High LOD Terrain Model

Low LOD model with Splot texture

The scene on the following page clearly shows the balance point in the trade space. This is a ground view of a database with a clip texture resolution of .75 meters. In the near ground the image is soft due to not enough resolution. But take note of how the shadows under the trees visually tie them to the ground.



High LOD Terrain Model

Low LOD model with Splot texture

10 Conclusion

There are many trades in the design and implementation of databases for real-time simulation. Silicon Graphics Infinite Reality (iR) computer system's swing the pendulum decisively toward pixel throughput. Splot is a technique which takes advantage of the iR's ability to manage and render giga-bytes of texture. Much richer environments are made possible by capturing the 2D features in imagery and leaving the polygon budget for modeling the 3D elements of the world.

Visual Database Construction Techniques:

Database Designs

Large Area Texture & Terrain ~ "above it all"

Integrated Feature Modeling ~ "stuck in the dirt"

Performance Management ~ "going both ways"

Integrated Feature Modeling

"stuck in the dirt"

Integrated Feature Modeling "stuck in the dirt"

Ground Vehicle and Life Form simulation database construction requires the successful transformation of digital vector features and digital terrain elevation data into an integrated, three dimensional, geospatial synthetic natural environment database.

Integrated Feature Modeling "stuck in the dirt"

Outline of topics

- Review the Requirements for Ground based Visual Simulation Synthetic Natural Environment Database Construction
- Review Digital Source data manipulation and refinement techniques
- Highlight Real-time Visual Simulation Synthetic Natural Environment Database Construction Techniques

Integrated Feature Modeling Requirements

Frame Rates

- 15Hz, 20Hz, 30Hz and 60Hz Image Update Rate

Sight Distance

- 1 meter to 20 kilometers

Fields of View

- 360 °Vision Blocks to 0.25 °magnified thermal sensors

Coverage

- Range from small 2x3km to large 500x700km areas

Highly detailed terrain and features

Networked with other simulations

Integrated Feature Modeling Requirements

Various Application within the Ground Domain

- Life Form
  - *Dismounted Infantry, Flight Deck Crew, Airport Personnel*
- Driver
  - *Motorcycle, Tank Driver Trainer, Automotive R&D*
- Gunnery
  - *Main Battle Tank, Artillery, ...*
- Combat/Maneuver
  - *Armor, Infantry, Artillery, ...*
- Specials
  - *Engineering, Breaching, Scraps, ...*

Integrated Feature Modeling Requirements

	Life Form	Driving	Gunnery	Maneuver
Frame Rate	30-60Hz	30-60Hz	25-30Hz	15-30Hz
Sight Distance	200m	2-4km	4-8km	20km
Field of View	180°	180°	0.25°-40°	0.25°-360°
Coverage	~2x3km	~10x10km	4x10km	500x700km
Detail	High	High	Corridor	High
Sensors	IR, NVG	NVG	IR	IR,NVG,Radar
Networked	Mix	No	No	Yes

Integrated Feature Modeling Source Data

Synthetic Nature Environment Database creation is essentially the process of sampling the real world, and then reconstructing a three-dimensional, geospatial model from the sampled data.

Integrated Feature Modeling Source Data

The problem is getting a good sample of the real world

- Digital source data is incomplete
  - *Contains to much of what you don't need*
  - *Not enough of what you do*
- Digital source data is inaccurate
  - *Collected from imagery - great*
  - *Collect from cartographic material - not so great*
- Digital source data is inconsistent
  - *Dependent on the collectors to be consistent*
  - *Collector over time*

Integrated Feature Modeling Typical Digital Source Data

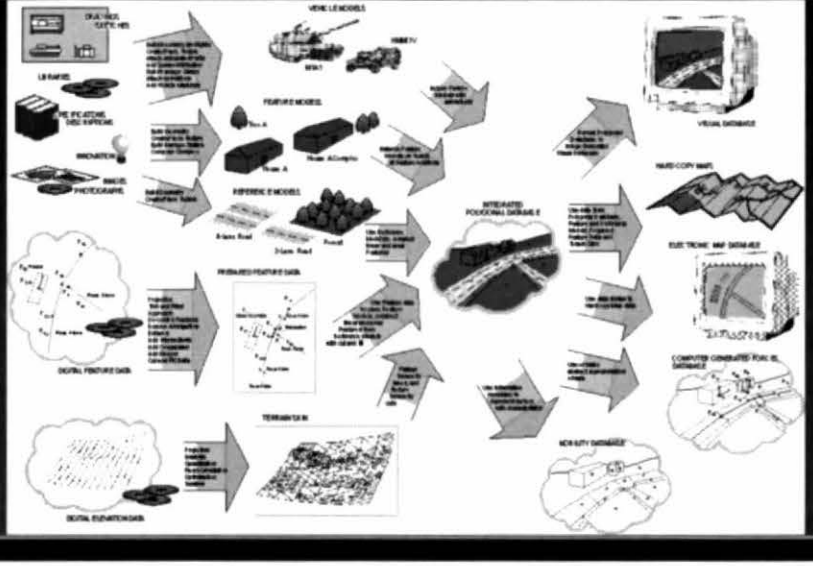
- Digital elevation models
  - *Equal spaced grid in a Geodetic Reference System*
  - *Collection is partly manual (stereoscopic pairs, site measurement, etc.)*
- Digital feature data
  - *Organized in classification schemes (themes, feature coding and attributes)*
  - *Fundamentally Two Dimensional (points, lines, areals)*
  - *Collection is manual (digitized from maps or photos)*

Integrated Feature Modeling
Source data collection

Digitized

Photo extracted

Integrated Feature Modeling
The Process



Integrated Feature Modeling
The Goal: 3D geospatial model

The Synthetic Natural Environment must be thought of as a whole

- The interactions between the various features and between the features and terrain must be carefully planned
- It is a matter of judicious planning and processing, not simply processing
- Refinement of the digital source data is the single most important step in achieving a useful three-dimensional, geospatial model
- Rules for construction must be established

Integrated Feature Modeling
Construction Rules

Feature to feature interaction must be explicit

- A bridge must exist whenever a road crosses river
- A bridge or causeway must exist whenever a road crosses a lake
- An overpass or intersection must exist whenever a road crosses another road
- Roads should connect to other roads

Feature to terrain interaction must be plausible

- Roads must be trafficable within the feature classification
- Three dimensional model must "rest" on the terrain

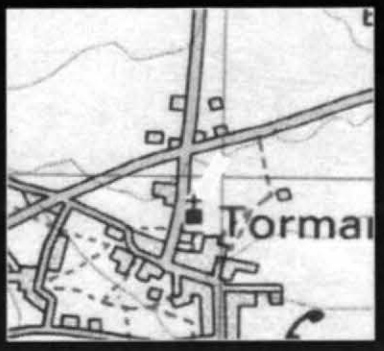
Integrated Feature Modeling
Source Data Refinements

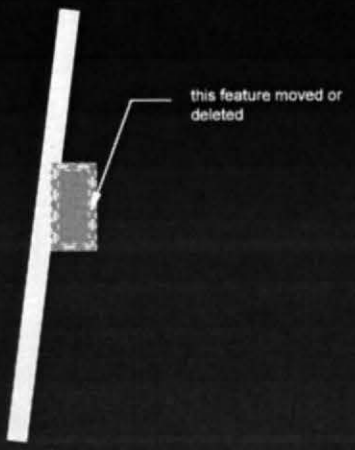
After the rules are established you can refine the data

- De-conflicting features
- Thinning features
- Generalizing features
- Smoothing features
- Editing errors
- Replacing features
- Synthesizing needed features
- Aggregating features
- De-conflicting terrain and features

Source Data Refinements
De-conflicting features

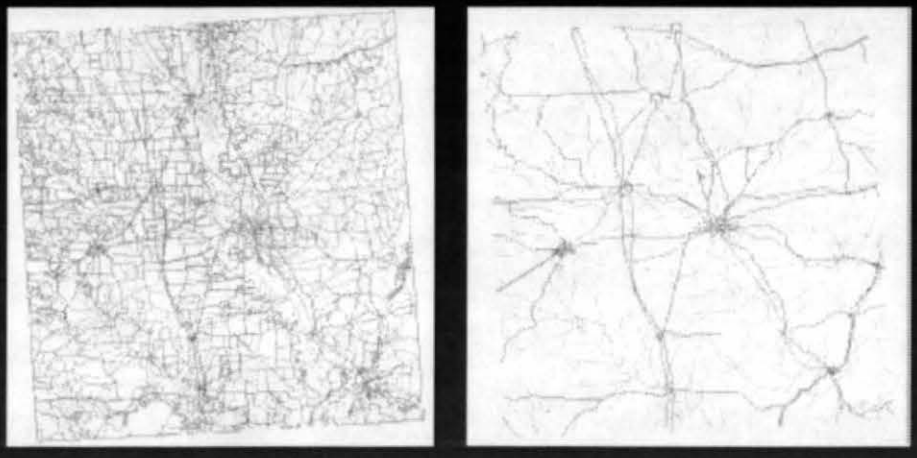
Conflicting feature is moved or deleted





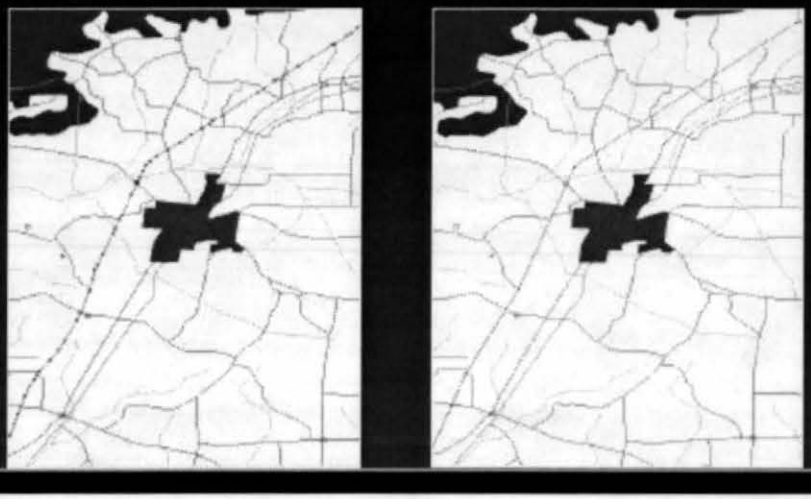
Source Data Refinements
Thinning features

Thinning reduces the number of features



Source Data Refinements
Generalizing features

Generalizing reduces feature detail

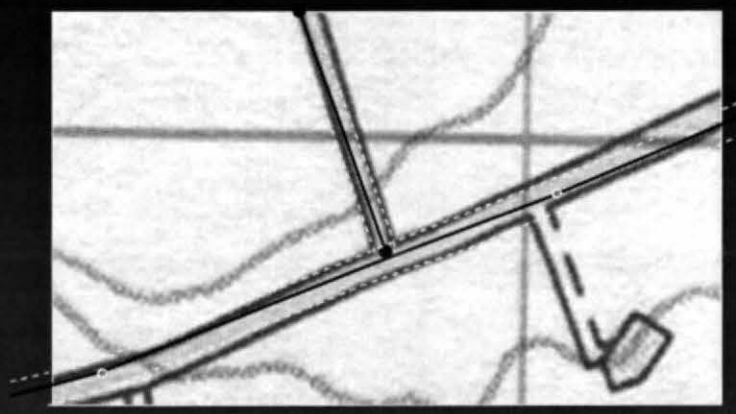


Source Data Refinements
Smoothing Features

Features are fit to spline curves and the points are redistributed

Source Data Refinements
Editing errors

Point is not coincident with segment



Source Data Refinements
Replacing features

The river feature areal is replaced by a linear featu

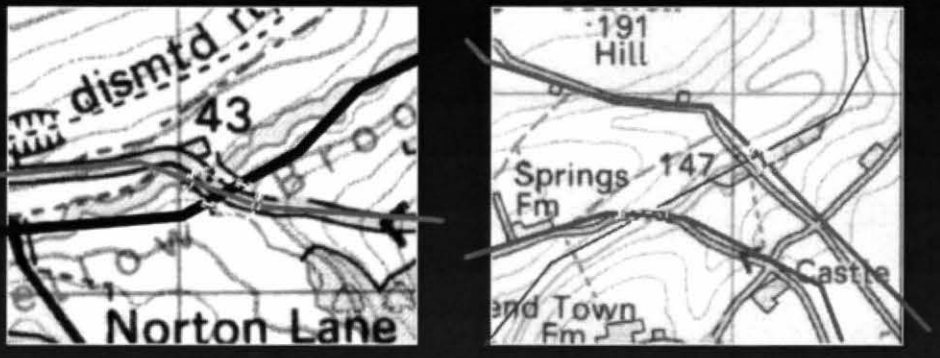
Source Data Refinements
Synthesizing Features

Data is lacking or non-existent

Data manipulation introduces anomalies

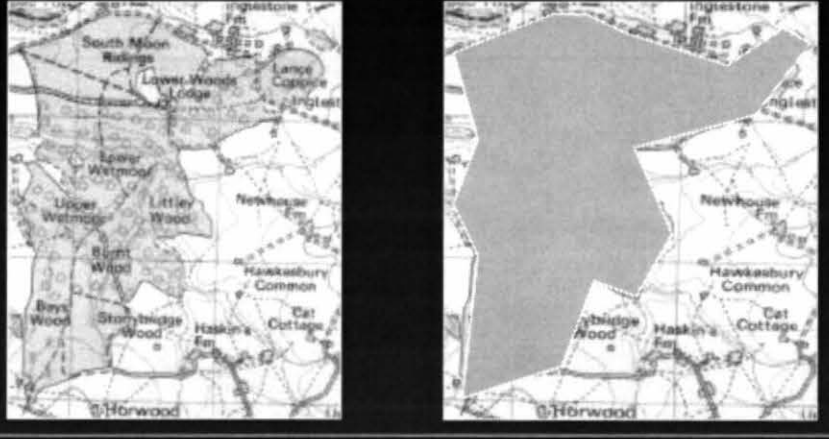
Source Data Refinements
Synthesizing Features

New data is created



Source Data Refinements
Aggregating features

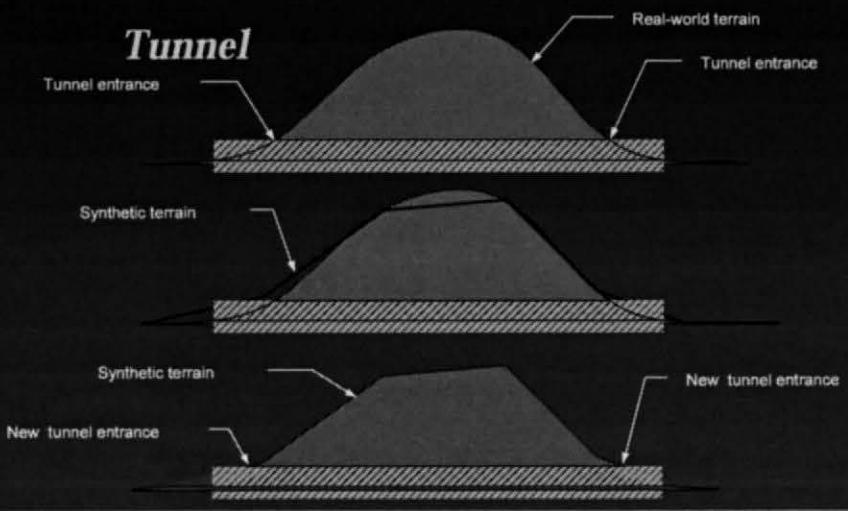
A wooded area is generalized and aggregated



Source Data Refinements
De-conflicting terrain and features

Tunnels interact with the terrain and other features

Source Data Refinements
De-conflicting terrain and features



Integrated Feature Modeling Database Construction Techniques

Real-time Visual Simulation Synthetic Environment Database Construction Techniques

- Distance and Relationship Cues
  - *Employing Texture and Geometry*
- Instancing Techniques
  - *Using Instancing for Point, Line and Areal features*
- Terrain Skin Optimization
  - *Optimized Adaptive Terrain Skin*
- Level of Detail Techniques
  - *Assuring scene integrity*
  -

Database Construction Techniques Distance and Relationship Cueing

Judging distance to a target is critical

- Assure consistency in size of objects
- Use texture frequency typical of the real world texture for terrain skin surface
- Do not blend textures on building when using low resolution texture maps
- Use monochrome noise pattern texture for detail
- Apply multiple texture maps for large dynamic viewing range
-
-

Database Construction Techniques Distance and Relationship Cueing

Judging spatial relationships of objects is critical

- Assure consistency in size of objects
- Place dirt spots under objects in open areas
- Add driveways and sidewalks to buildings
- Match texture detail on terrain skin facets and three dimensional models
- Use textures under features to portray shadows
- Place shadows under moving vehicles
-
-

Database Construction Techniques Instancing Techniques

The density of objects is key for Cover and Concealment

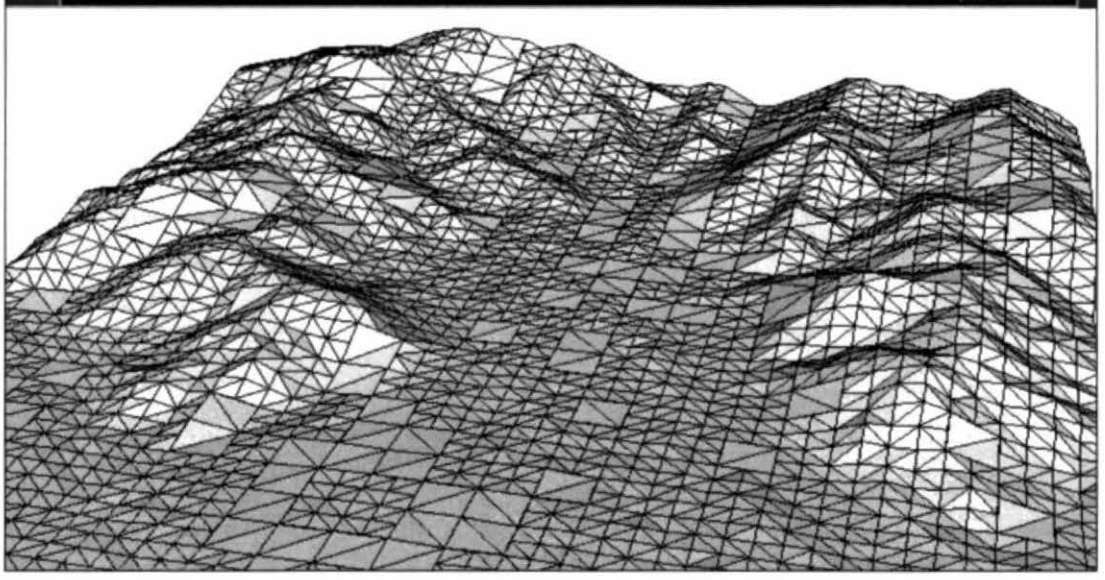
- Point Features
  - *Individual and Complex models for Landmarks*
- Line Features
  - *Reference models for automatic creation*
- Areal Features
  - *Basis Sets Usage*
    - Penetrable Forest
    - Texture roads in Rural and Industrial Built-up Area
  - *Algorithmic Orientation Selection*

Database Construction Techniques
Terrain Skin Optimization

Getting the most from a terrain skin

- Terrain skin facets focussed in areas where vehicles travel
- Terrain skin simplified where addition decomposition provides no additional detail
- Right triangles of varying size
- Similar goals to the Integrated Triangulated Irregular Networks, but retains the optimization of grids

Terrain Skin Optimization
Optimized Adaptive Terrain Skin



Database Construction Techniques
Level of Detail Techniques

Assuring Scene Integrity

- Must match the transition ranges and fade ranges of terrain, three dimensional features and moving models
- Moving vehicles transition ranges and fade ranges determined by Johnson Criteria
- Terrain simplification does not occur until the vehicles fade from the scene
- Three dimensional features providing cover and concealment are not faded until the vehicles fade from the scene

Non-Linear Distortion Correction

Warped Views of Reality

Stephen H. Gersuk
Paradigm Simulation, Inc.
shg@ParadigmSim.com

What is Geometric Distortion?

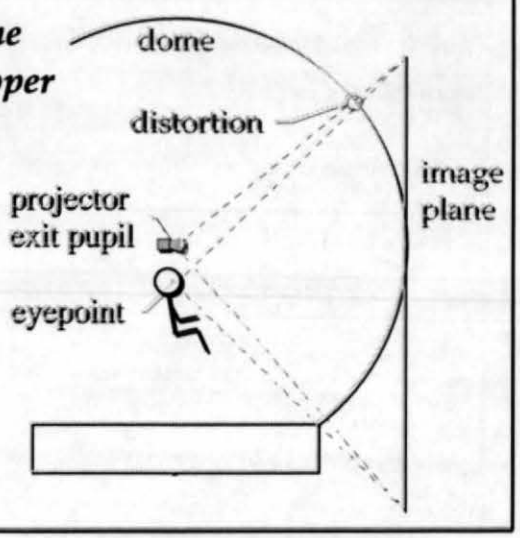
... displacement of scene elements from their proper perspective positions

... a result of:

- screen curvature
- lens distortion
- projection geometry

... specified as either

- percent of screen height
- angular displacement



Display Systems

Boeing (MDHC) SOPS

- 120° x 90° IFOV from low-res background plus 40° x 30° coaxial AOI
- head-slaved: 300° x 180° TFOR

Boeing (MDTS)

- two projectors illuminate forward and aft hemispheres
- fixed AOI
- "hidden hole" projection optics

Other systems

- E&S - many multi-channel domes, including fixed, head- and target-slaved projectors
- Raytheon (Hughes)
- Sogitec

How is it Corrected?

E&S "Non-Linear Image Mapping"

- Subdivides of polygons based on angular extent

Lockheed Martin "Comprehensive Distortion Correction"

- Similar to method described here

Paradigm Non-Linear Distortion Correction

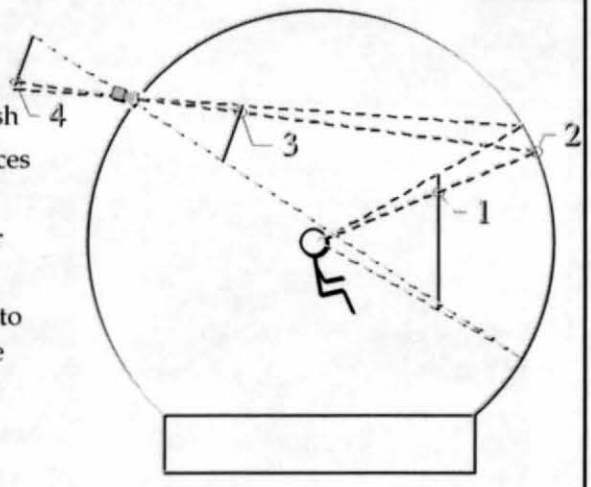
- Two-step process
  - *Ray trace performed in App*
  - *Image warping performed in Draw*
- Static or dynamic (for fixed or slaved projection optics)

Ray Trace Process

App:

1. Subdivide channel image plane into mesh
2. Ray-trace mesh vertices to dome surface
3. Ray trace to projector image plane
4. Map through lens onto projector object plane

Performed once at init or as required

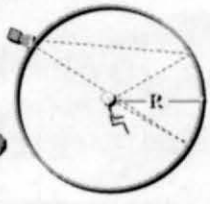


Dome Shapes

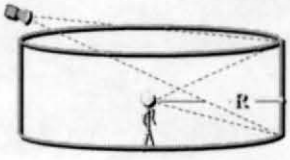
Common surfaces are all quadrics:

$$x^2 + y^2 + gz^2 - d = 0$$

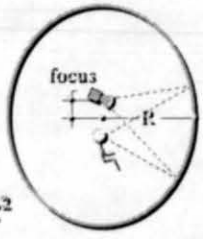
Parameters g & R define different shapes



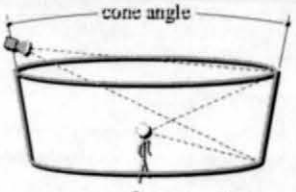
$g = 1 \quad d = R^2$



$g = 0 \quad d = R^2$



$$g = \frac{1}{1 + (focus/R)^2} \quad d = R^2$$



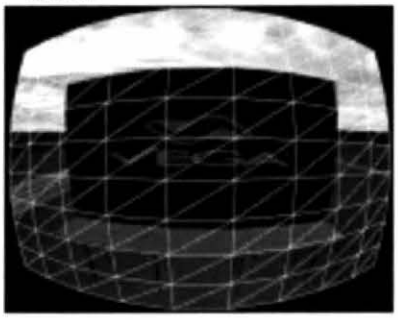
$g = -\tan^2 \kappa/2 \quad d = 0$

Warping Process

Draw:

- Render perspective channel(s)
- Copy image(s) from frame buffer to texture memory

- Render mesh(es) in ortho output channel
- Apply computed texture(s) to mesh



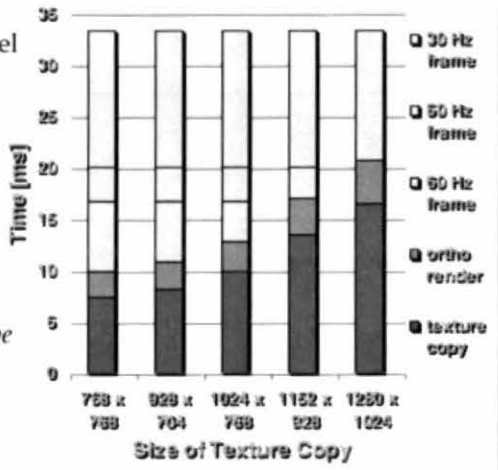
The Effect on Performance

App:

- Ray trace ≈ 2 - 3 ms / channel

Draw:

- Copy rate ≈ 80 M texels/s (`glCopyTexSubImage2DEXT`)
- Ortho channel ≈ 3 - 4 ms
- Perspective channels get what's left
- *Dynamic resizing of perspective channels offers highly-levered load management*



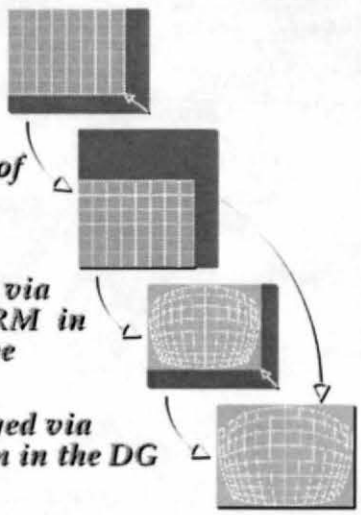
Dynamic Resizing

1. nominal perspective image, minified by application

2. active area copied to corner of allocated 2^s x 2^m texture; (u,v)'s remapped to active area

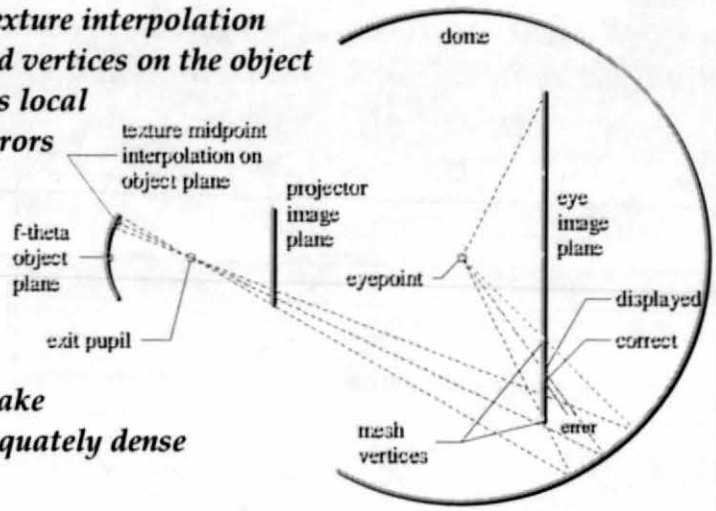
3. texture applied to mesh via bilinear interpolation in RM in ortho channel; may also be dynamically minified

4. active area displayed via bilinear interpolation in the DG (SGI only)



Interpolation Error

Problem: Texture interpolation between grid vertices on the object plane causes local geometry errors (e.g., kinks in roads)



Solution: make the grid adequately dense

Alignment

Sources of error:

- Dome
  - *radius, surface irregularities*
- Raster
  - *horizontal & vertical size & position*
  - *rotation*
  - *geometric distortion*
- Optics
  - *exit pupil position*
  - *distortions beyond nominal "f-θ" prescription*

Method

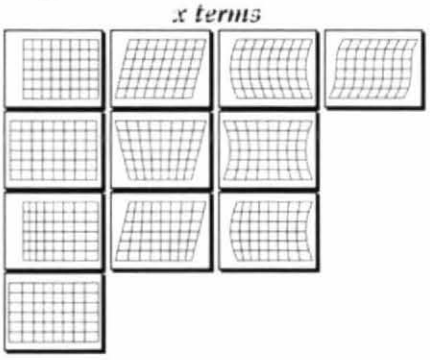
- Modify the lens map via two sixth-order polynomials (xOfst is shown, zOfst is symmetric):

$$xofst = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^5 \end{bmatrix}^T \begin{bmatrix} bx_{00} & bx_{01} & \cdots & bx_{05} \\ bx_{10} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ bx_{50} & \cdots & \cdots & bx_{55} \end{bmatrix} \begin{bmatrix} 1 \\ z \\ \vdots \\ z^5 \end{bmatrix}$$

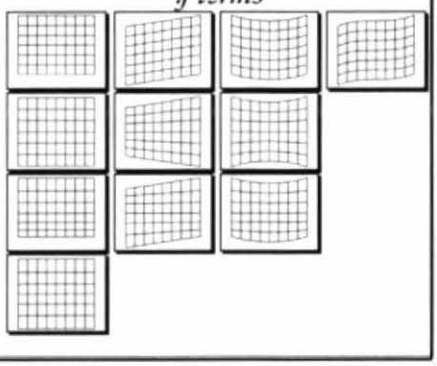
Polynomial Correction

Coefficients correspond to algebraic raster shaping adjustments on a CRT projector (shown), plus higher-order terms

x terms

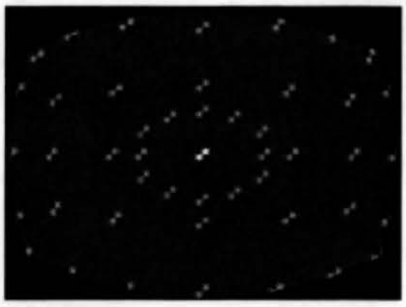


y terms



Alignment Procedure

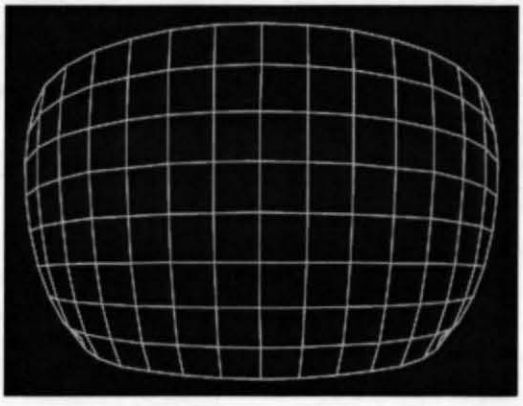
1. *Render target at nominal (az, el) of each LED*
2. *Center targets over LEDs*
3. *Compute polynomial coefficients for least-squares fit to displacements on object plane*
4. *Use x and z polynomials to correct lens maps*



Powerful Correction

Just the 11 terms shown previously could nearly correct (green) the SOPS raster (red)

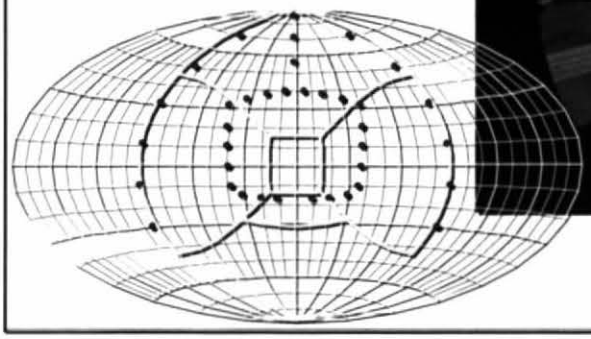
With 27 additional terms available, highly accurate fits are possible to any systemic distortion



Channel Mosaics

Multiple channels may be used to tile the field of view

Channels may overlap or no



Real-Time Graphics For Visual Simulation

Displaying the Results - Creative Uses of Bitplanes

Elizabeth Smith

Paradigm Simulation, Inc.

Real-Time Graphics For Visual Simulation

Displaying the Results - the Framebuffer

- Partitions of the framebuffer and pixel depth
- Framebuffer operations for Visual Simulation applications
- Performance considerations

Real-Time Graphics For Visual Simulation

Buffers in the FrameBuffer - Pixel Depth

- Color Buffers - RGBA / ColorIndex - Front / Back
- Depth Buffer - Z
- Stereo - Left / Right
- Stencil
- Overlay / Underlay
- Multisampling - Antialiasing
- Accumulation Buffers
- Off-Screen Rendering
- Miscellaneous - Auxiliary

Real-Time Graphics For Visual Simulation

Example: Fast Landing Light Lobes

Real-Time Graphics For Visual Simulation

Example: Fast Landing Light Lobes

Real-Time Graphics For Visual Simulation

Example: Fast Landing Light Lobes

- Problems with spotlights
  - *Work best with many small polygons.*
  - *Expensive.*

Real-Time Graphics For Visual Simulation

Example: Fast Landing Light Lobes

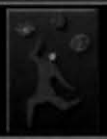
- Problems with projected textures
  - *Multipass.*
  - *Black fog for attenuation - not good at twilight*
  - *Uses texture memory.*
  - *Problems with cumulative lobes.*

Real-Time Graphics For Visual Simulation

Example: Fast Landing Light Lobes

- Use destination alpha - the "A" in RGBA
- Clear screen with alpha darkness factor
- Draw a transparent mesh for the conic section of each lobe
- Draw the meshes into the alpha planes only
- Have blending set to accumulate sourceAlpha and destinationAlpha of lobes

Real-Time Graphics For Visual Simulation



Example: Fast Landing Light Lobes

- Mask out the alpha planes
- Draw the entire scene, except for emitting polygons, blending with the destination (saved) alpha: srcR/G/B * destAlpha
- Use an infinite light (cheap) at the light lobe source whose intensity is 1 - time-of-day; where time-of-day is between 0.0 and 1.0

Real-Time Graphics For Visual Simulation

Example: Fast Landing Light Lobes

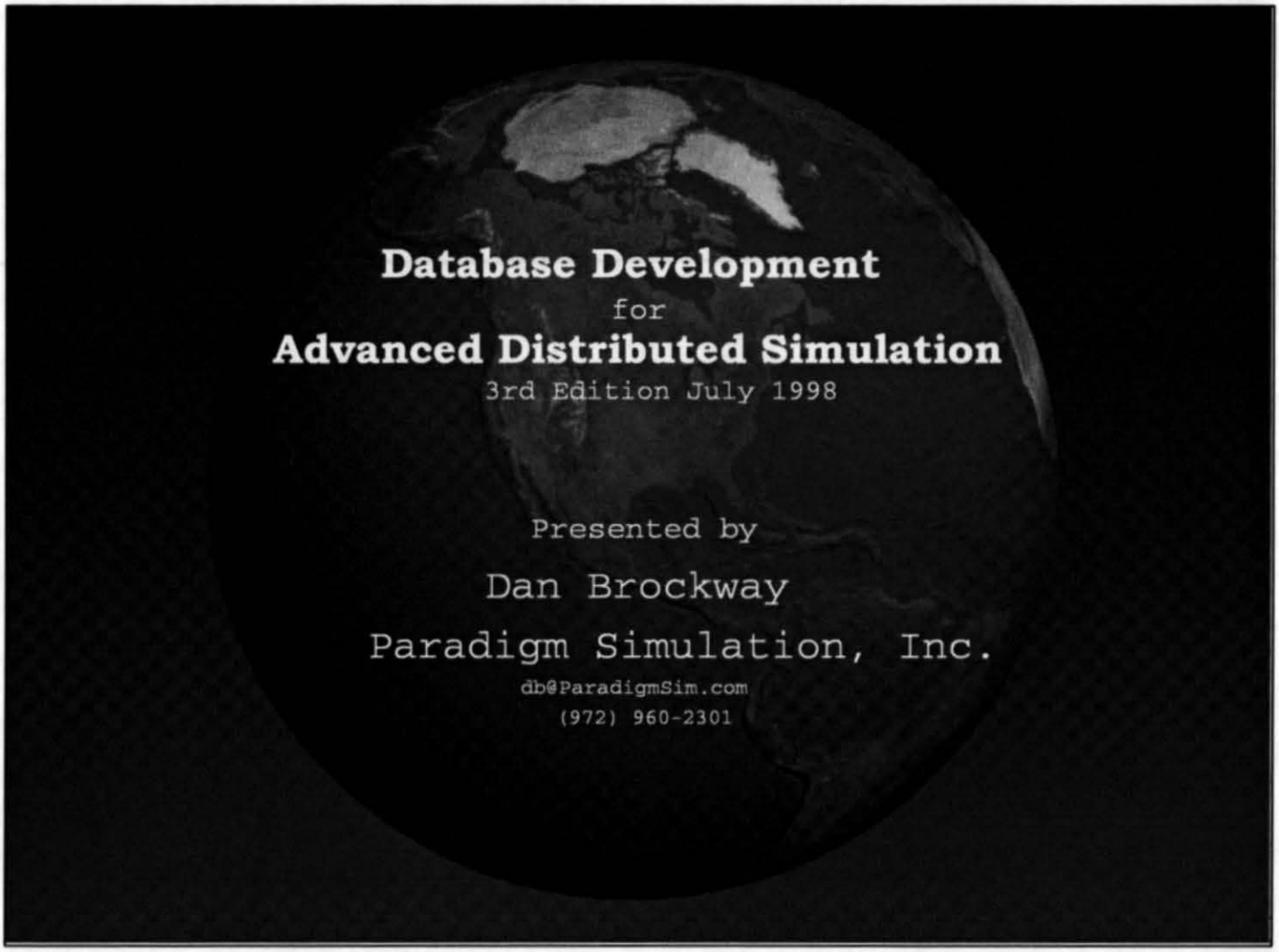
- Requires an R G B A visual
- Penalty for reading destination alpha
- More polygons to draw - the triangle meshes
- Looks best when antialiasing is available
- Clear pass needed
- Must treat emitting polygons special
- Source of light must be near observer

Real-Time Graphics For Visual Simulation

Example: Fast Landing Light Lobes

Alternate Approach:

- Needs no destination alpha
- Draw lobe to stencil plane after scene, blending with source alpha values of the lobe triangles
- Darken scene, except for stenciled lobe area, with a screen-size polygon having alpha equal to normalized time of day (0.0. to 1.0)
- Lobes cannot accumulate



Database Development
for
Advanced Distributed Simulation

3rd Edition July 1998

Presented by

Dan Brockway

Paradigm Simulation, Inc.

db@ParadigmSim.com

(972) 960-2301

Database Development
for
Advanced Distributed Simulation
3rd Edition July 1998

Presented by

Dan Brockway

Paradigm Simulation, Inc

db@ParadigmSim com

(972) 960 2301

1st edition developed at SAIC on contract to the
Advanced Research Projects Agency
Under the War Breaker
SEM Contract

Page 2

Contents

Introduction

- Distributed Simulation and the Environment

Part One The Reference Model

- A data flow

Part Two Operations on Data

- The Synthetic World

Part Three Communicating the Product

- Define the Environment
- Define Thematic & Geometric Data Models
- Map the Environment onto Data Models

Part Four Data Exchange

- Product Specifications vs Format Specifications

Introduction

Distributed Simulation and the Environment

Page 4

Advanced Distributed Simulation

Advanced Distributed Simulation is not a single thing or model that can be rigorously specified and built, but rather a suite of constantly improving tools and techniques for assembling dynamic and disparate federations of models and simulations and real platforms and real systems of all types and sizes that *interoperate* together.

Source Jack Thorpe

Page 5

Over the past year Distributed Simulation has begun a new evolution based on the DMSO High Level Architecture effort

Application Domains

Applications include

- platform (driver) training
- crew/unit training
- mission rehearsal
- tactics development/evaluation
- testing effectiveness of weapons systems at the conceptual, developmental, prototype, and operational test phases
- force structuring analysis
- and others

Source DIS Vision Document

Page 6

Database Requirements

- **Air to air**
  - **Large geographic areas**
  - **Little or no elevation**
- **Air to ground**
- **Navigation, landing and ground handeling**
- **Close air support**
- **Infantry**
- **Marine amfibious assult**
- **Mine/countermine**
- **Combat engineering**
- **Team tactical training**
- **Urban combat**

The Synthetic Environment Challenge

- **Each modeling and simulation application has a different set of requirements for environment elements and their representations within the distributed simulation.**
- **Achieving the goal of a distributed simulation environment where new players can gain from, and contribute to, a rich battlefield environment requires agreement on how the world is modeled.**
- **The representations used within synthetic worlds must be documented so that newcomers can assess the validity of play within a given battlefield.**

Page 8

Why its a Hard Problem

The Environment is a continuous and changing place, there are few discrete boundaries.

All modeling and simulation tasks require making tradeoffs to achieve appropriate abstractions of the world so that the simulations can function within current computer technology.

Finding common abstractions which serve disparate simulations is a formidable task.

Toward a Solution

Expose the Process

- **The World Reference Model as a generic data flow with descriptions of the needed data processing**
- **The Synthetic World as the integration of data which is shared by all distributed participants**

Communicate the Solution

- **Define a semantic to describe the environment**
- **Define Thematic & Geometric Data Models**
- **Express the Synthetic World as a mapping of environment elements onto the data models**

Exchange the Data

- **Understand Product Specifications vs Format Specifications**
- **SEDRIS**

Page 10

Benefit of the Approach

This approach makes it possible to define a synthetic world by its content and the data model representations it supports without prescribing a modeling solution

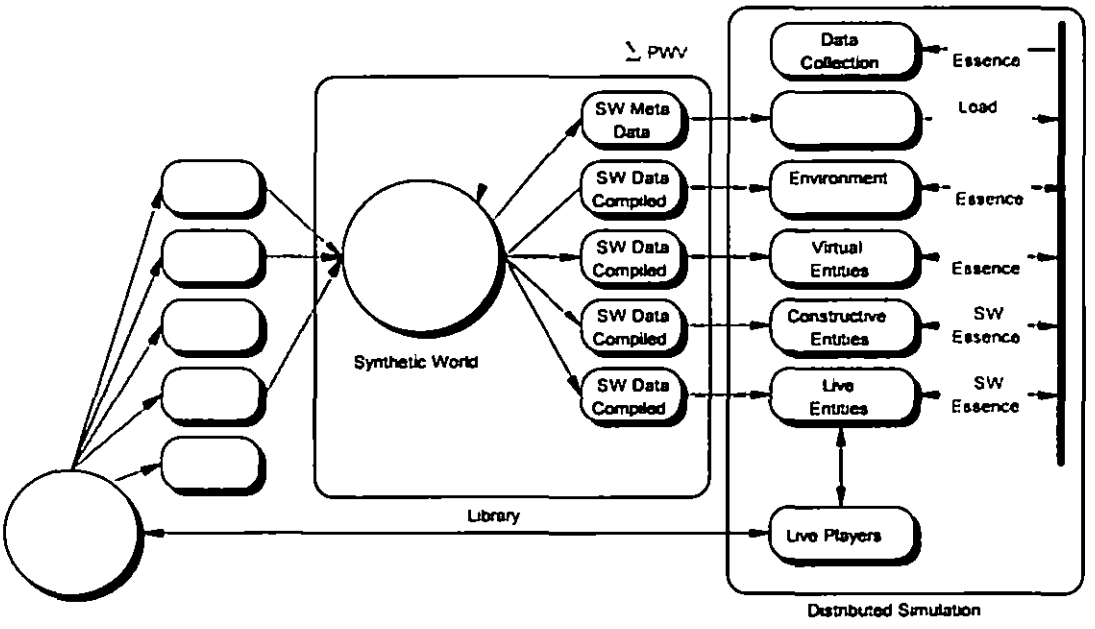
Part One

The Reference Model

- **The World Reference Model was born out of the need to understand the types of processes required to shape source data in preparation for simulator initialization**
- **The WRM was developed at SAIC to address the needs of the War Breaker program**
- **The model provides a useful exposition of the process of database creation**

Page 11

The World Reference Model



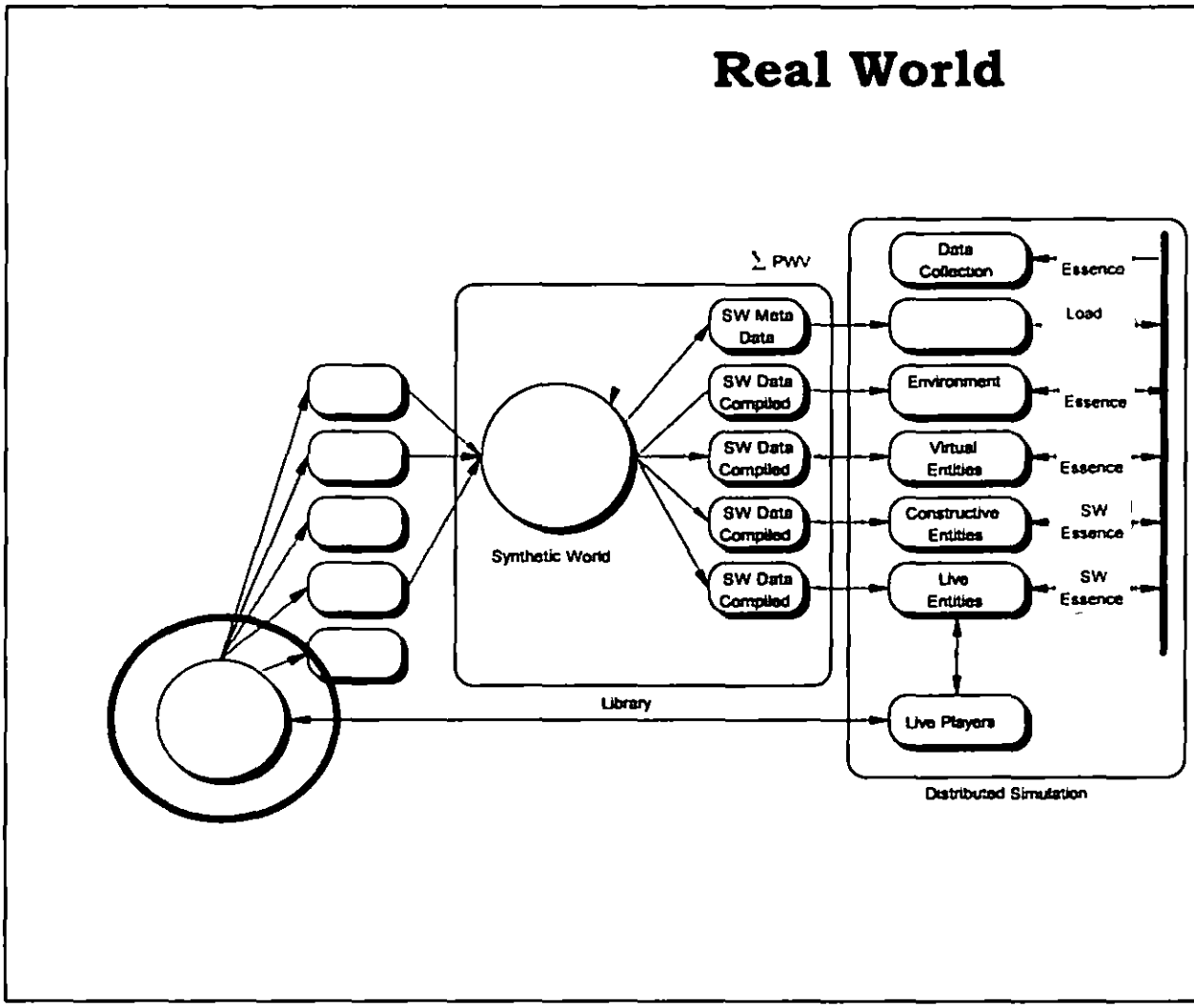
Page 12

The World Reference Model (WRM) is a model which defines the context around the Synthetic World

It shows where source is gathered and refined in a Thematic Data Model then coalesces into a machine independent Geometric Data Model which is then compiled into simulator specific databases The World Reference Model is grounded on the premise that Advanced Distributed Simulations will be truly heterogeneous and completely dynamic The central point identified by the World Reference Model is the need for a simulator independent data model from which simulator optimized data can be unambiguously produced

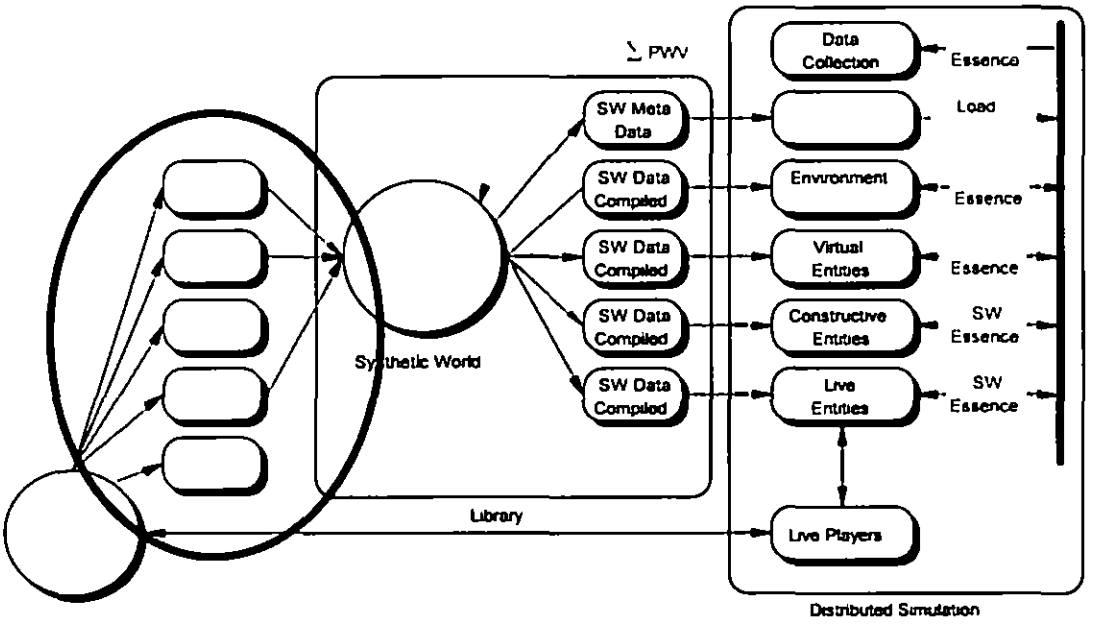
The World Reference Model (WRM) is a framework within which to discuss

- the flow and refinement of source data into simulator databases
- the desire to share data at various points along the refinement path
- the use of coordinate systems projections and conversions
- the use of attribution standards
- the effects of simulation/simulator requirements on data
- the issues associated with dynamic terrain



The World Reference Model considers the Real World as the geographic area to be simulated The size of that area is not constrained by the WRM it could be a 100km x 100km test range a road intersection or the entire globe

Sources



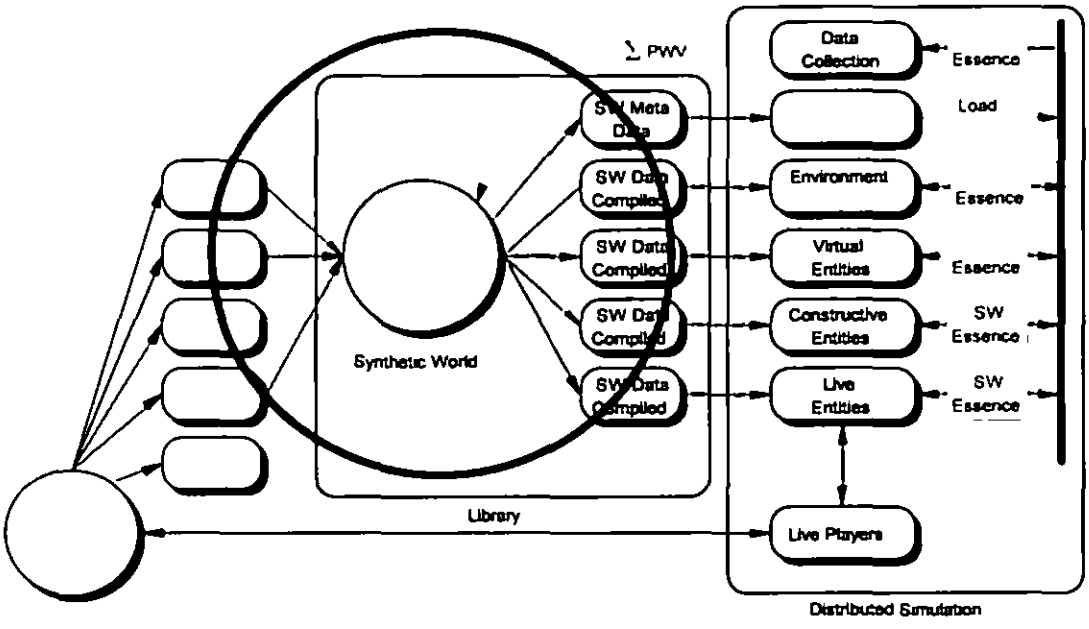
Page 14

Sources (digital map data and imagery products) are measurement abstractions of the real world They represent the measurements made from the real world and are the source from which a Synthetic World is made

Source data is often conveyed as thematic layers where measurements of a single type are stored together ignoring their relationship with other kinds of data For example Digital Terrain Elevation Data (DTED) and Digital Feature Attribute Data (DFAD) are separate layers even though they may in fact cover the same geographic area

Common sources include the Defense Mapping Agency (DMA) the United States Geological Survey (USGS) SPOT Image Corporation imagery and LANDSAT

Synthetic World



Page 15

The Synthetic World is an important focus in the World Reference Model It is the point at which source data coalesces into a single set of integrated databases which adequately represent the spatial context for Distributed Simulation The Synthetic World is where we re-integrate the thematic layers and resolve their conflicts It is that unified set of geometric data that represents the ultimate definition of simulation truth

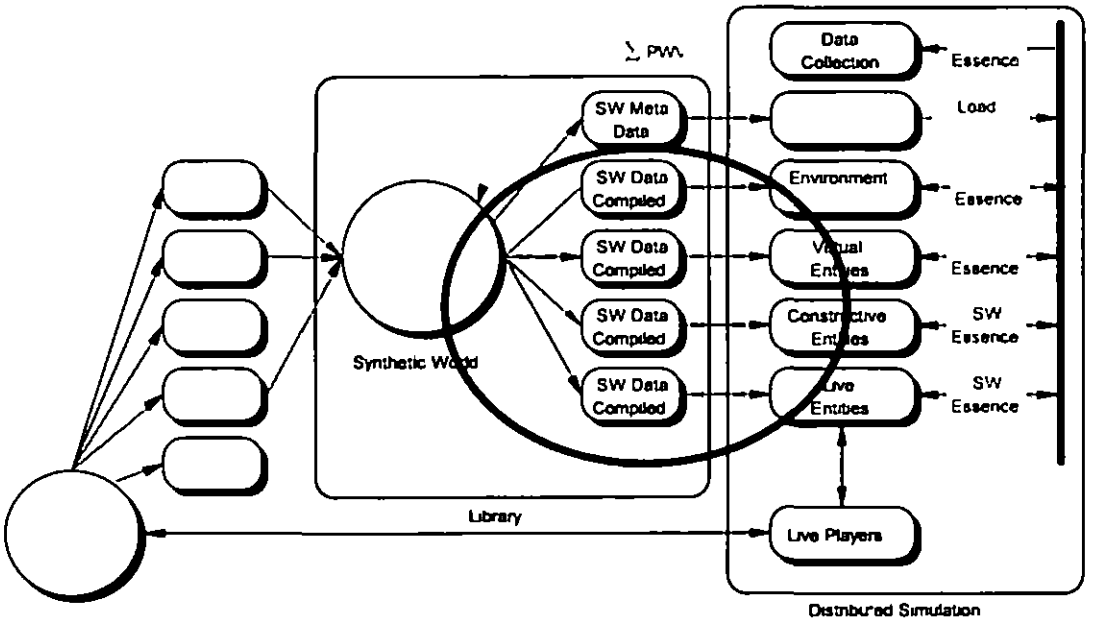
A Synthetic World must be designed with knowledge of the simulation technology to be applied and the objectives of the Distributed Simulations for which this spatial representation will be used

Because the goal is to address heterogeneous simulation environments our Synthetic World is not defined in a simulator-optimized format We want to capture the truest possible description of the environment within the constraints of available sources and customer requirements

Correlation against the source should be measured to understand how much error has been introduced into the process by integrating data

this measurement can be made against each data source The essential correlation will depend on the importance of each source to the end application For example if road shape is critical then correlation with the road source will be critical

Simulator Databases

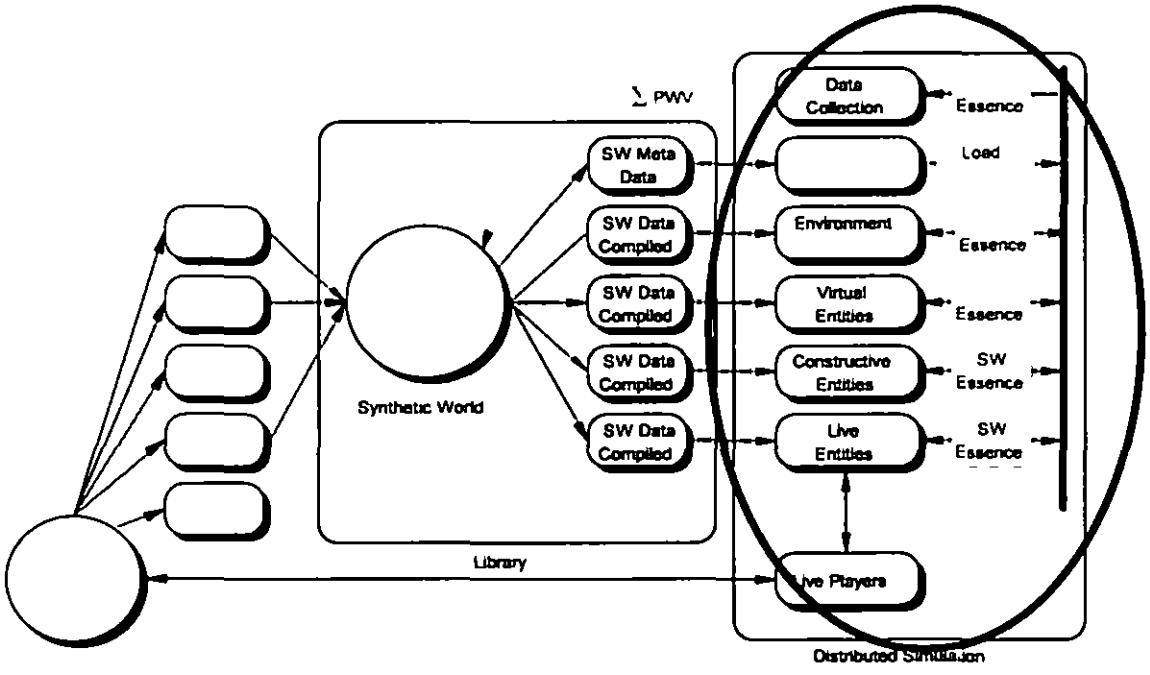


Page 16

Simulator databases are extractions of data from the Synthetic World formatted to the application and performance criteria of specific simulators

It is at this point where optimized configurations of data from the Synthetic World are packaged for direct use within a specific simulator

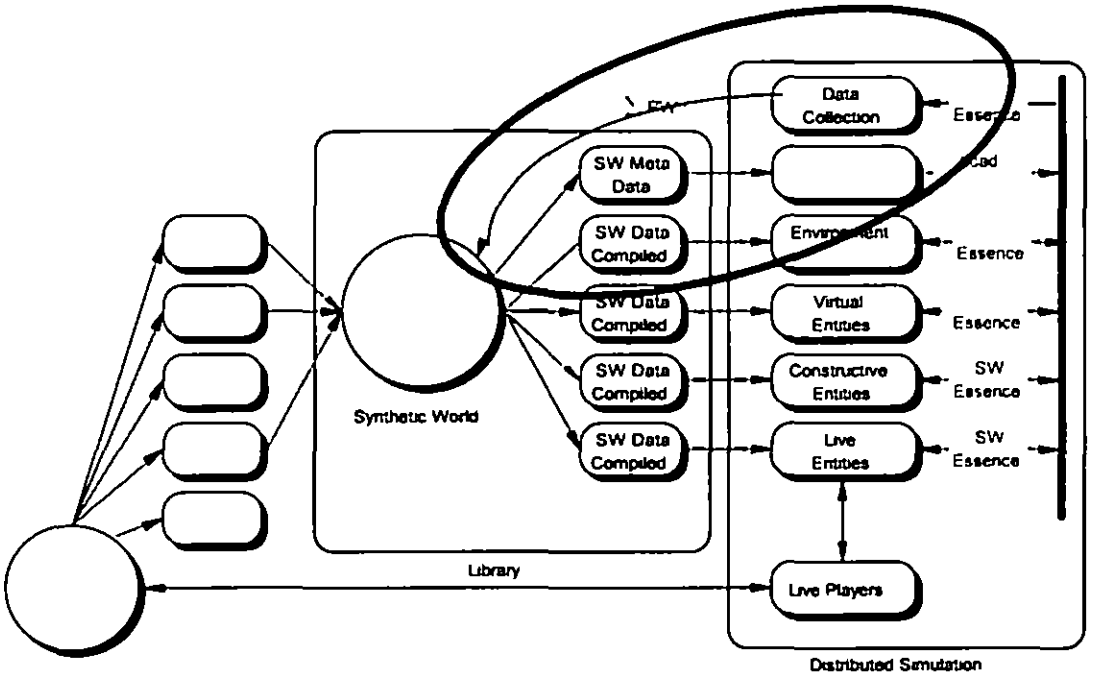
Distributed Simulations



Page 17

A Distributed Simulation is the inter-operation of Virtual Constructive and/or Live simulators to address a specific issue A Distributed Simulation is composed of only the simulators which can present the appropriate views of the Synthetic World (and the Real World in the Live case) to the users of the system

Start Up & Shut Down



Page 18

Start Up

In theory a simulation management function would be used to instruct each of the entity simulations which initialization data to load For example in a simulation where environment entities were modeling the atmosphere the Environment Entity would be instructed to load the cloud data and the Virtual Entities would be instructed to load a configuration which did not include clouds

Shut Down

Again in theory a data collection function could gather up all the changes to the world and create a snapshot (check point) This data would be returned to the library as a new configuration of the world

The plausibility of these theories is dependent on the dynamic configurability of the simulators and a responsive network delivery mechanism

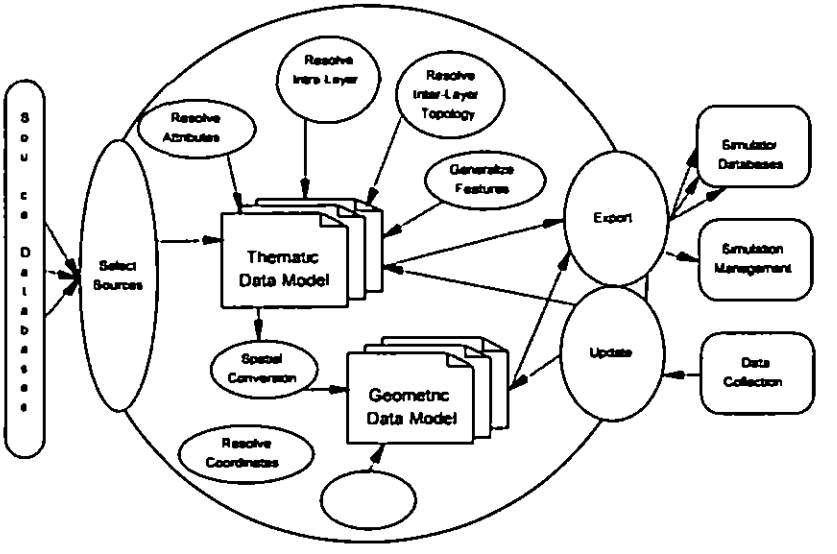
Part Two

Operations on Data

- **A description of the operations which must be preformed on data to prepare a Synthetic World**

Page 18

Synthetic World



Page 19

The Synthetic World is organized around two data models and the processes which operate on those models

Thematic Data Model

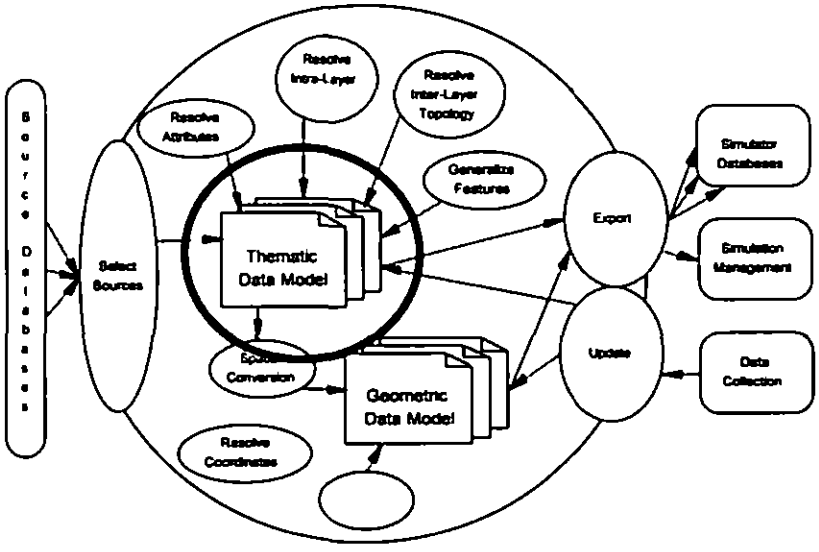
The first model is the Thematic Data Model which closely corresponds to the source data The data is thematic because it consists of a single specific type of information as measured from the Real World Each theme is delivered either as a separate product such as DTED DFAD and imagery or as a multi-theme product such as VITD While thematic layers together describe the attributes and planametric geometry of an area we do not yet consider them a 3D geometric description The reason is simple thematic data comes from maps and digital map products which were never intended to convey the detail necessary to build man in-the-loop simulations

Geometric Data Model

The second model the Geometric Data Model goes the extra step Its definition contains all the data necessary to produce a geometric model which can be rendered in today s computer image generators and a topologic model which can support computer generated forces

The World Reference Model **distinguishes** between the **Geometric Data Model** of the Synthetic World and the **extracted databases** used by simulators The geometric Data Model describes the world in as much detail as agreed upon, but it may not be stored in an optimized format Simulator databases on the other hand, are formatted and optimized to meet each image generator's (or Computer Generated Forces system's) ability to actually load and use the data In many cases data is lost (or gained) in the compilation of a simulator database

Thematic Data Model



Page 20

Thematic Data Model

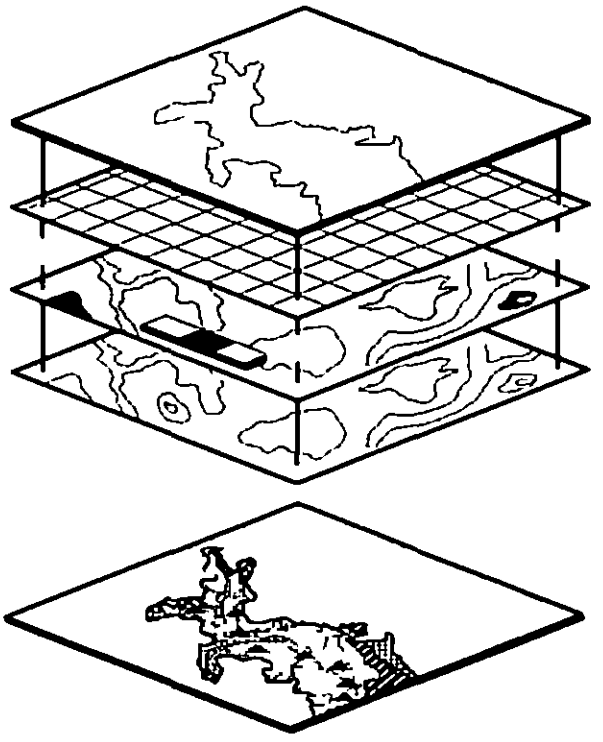
The Thematic Data Model is the set of thematic layers which define the environment A thematic layer is a set of data which covers the geographic area and provides at least one attribute Examples of thematic layers include the Defense Mapping Agency's (DMA) Digital Terrain Elevation Data (DTED) which provides an elevation attribute on a geodetic grid DMA s Interim Terrain Data (ITD) which provides six thematic layers (vegetation lines of communication slope obstacles surface drainage surface material) DMA s Digital Feature Analysis Data (DFAD) and Electro-Optical or Multi-Spectral satellite imagery

The content of the Thematic Data Model is dependent on the intended use of the Synthetic World The WRM makes no attempt to define which data are necessary for any given Synthetic World However the opportunity does exist at the DIS community level to define categories of Synthetic Worlds based on the types of thematic layers that programs are using to solve specific simulation objectives

Thematic Data Model Example

The following layers define the example Thematic Data Model

- **Point features**
- **Line features**
- **Land Cover Features**
- **Soils**
- **Surface Shape**
- **Land Cover Imagery**
- **Cloud Imagery**

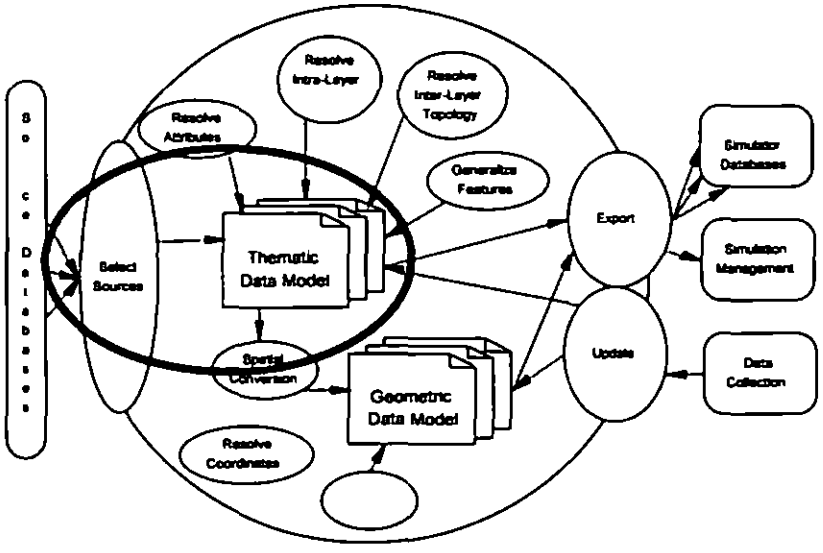


Page 21

A Thematic Data Model can be defined by any number of layers

This specific set of layers is used throughout this presentation for illustration

Select Sources



Page 22

Select Sources

The character and quality of a Synthetic World is defined by the data selected as source The builder of a Synthetic World carries the responsibility of maintaining documentation for the provenance of all data use

Sources Examples

DTED

- **Elevation grid** -> **Surface Shape**

ITD

- **Surface Materials** -> **Soil Area Features**
- **Vegetation** -> **Land Cover Features**
- **Hydrography** -> **Line Features, Land Cover Features**
- **Transportation** -> **Point Features, Line Features**
- **Obstacles** -> **Point Features, Line Features**
- **Slopes** -> **Not used**

Spot Image

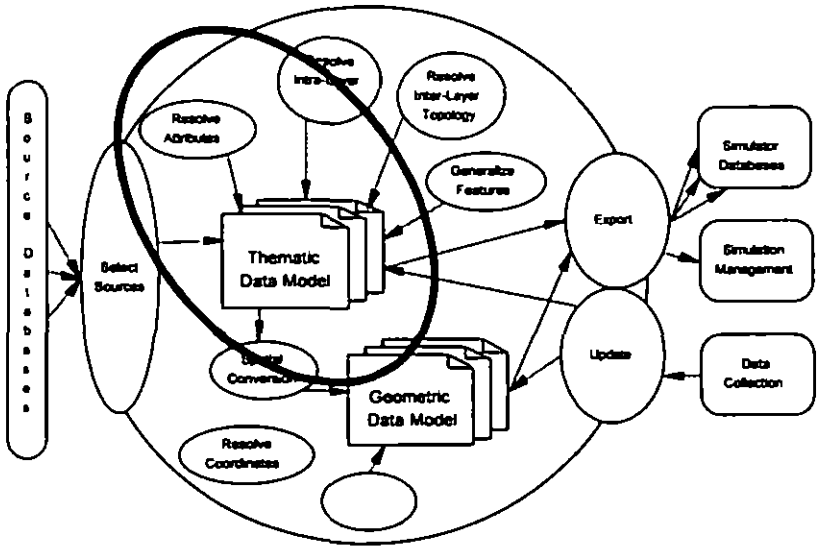
- **10m Panchromatic** -> **Land Cover Imagery**

WINDS

- **Liquid Water Content** -> **Cloud Imagery**

Page 23

Resolve Attributes



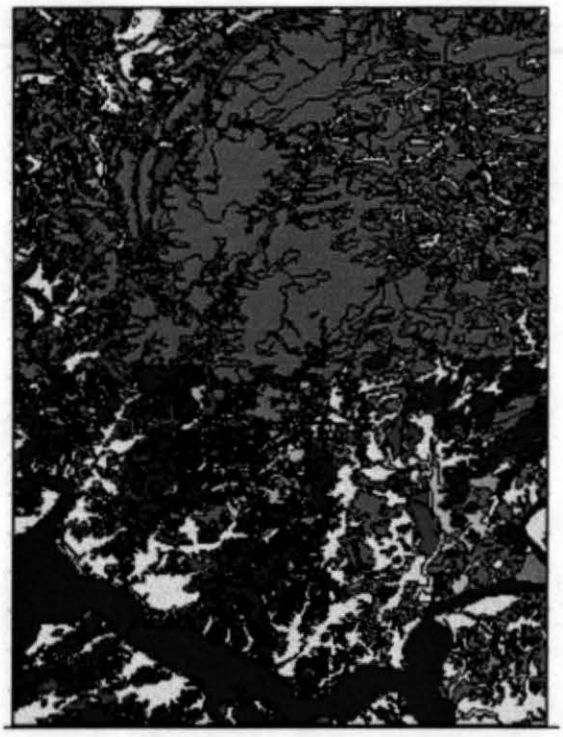
Page 24

Resolve Attributes

The design of a Synthetic World must include an attribution schema which defines how elements of the environment are encoded Although DMA internally uses a Feature and Attribute Code Catalogue (FACC) which is based on DIGEST for encoding features each product conforms to a separate attribution schema For example DFAD prescribes 14 surface material types (i e concrete grass etc) and 36 feature identifications (i e church road etc) Within the Synthetic World a single attribution scheme must be defined and all data made to conform to that standard

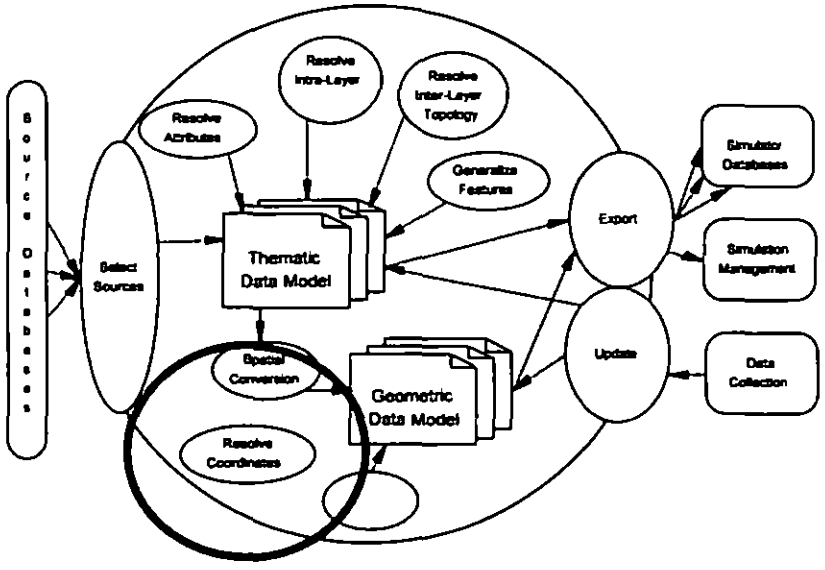
In this "resolve attributes process inconsistencies within the attribution scheme must be identified and resolved Consider for example a road network derived from two DMA map sheets It is possible that the analyst who digitized the left sheet considered dirt trails to be tertiary roads and the analyst who digitized the right sheet considered dirt trails as secondary roads In a Synthetic World which only requires secondary roads this situation produces excessive data in the right sheet When possible re-attributing or deleting the trails would be done in this process

Resolve Attributes Examples



Page 25

Resolve Coordinates



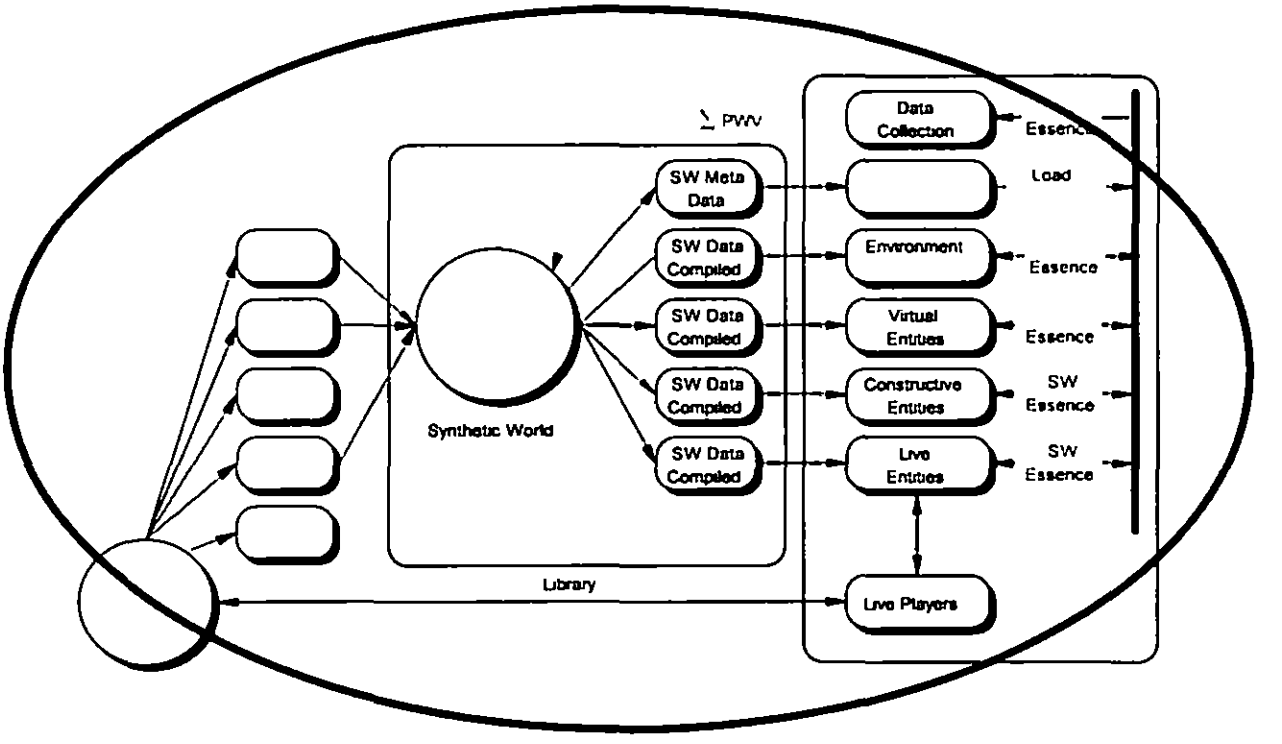
Page 26

Resolve Coordinates

The WRM asserts that within the Synthetic World all data must be referenced to a single coordinate reference system Although the choice of systems can be made on a case by case basis mathematical rigor must be applied to ensure that the data is not unduly degraded and that conversion to external systems is possible

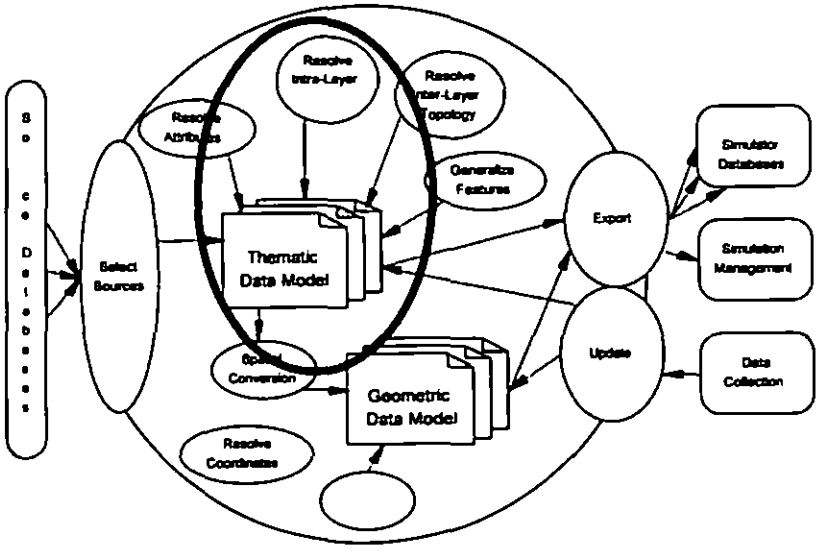
The issue of coordinate systems and projections needs to be addressed against the data flow diagram to ensure that no unnecessary transformations occur

Resolve Coordinates – a System Issue



Page 27

Resolve Intra-Layer Topology

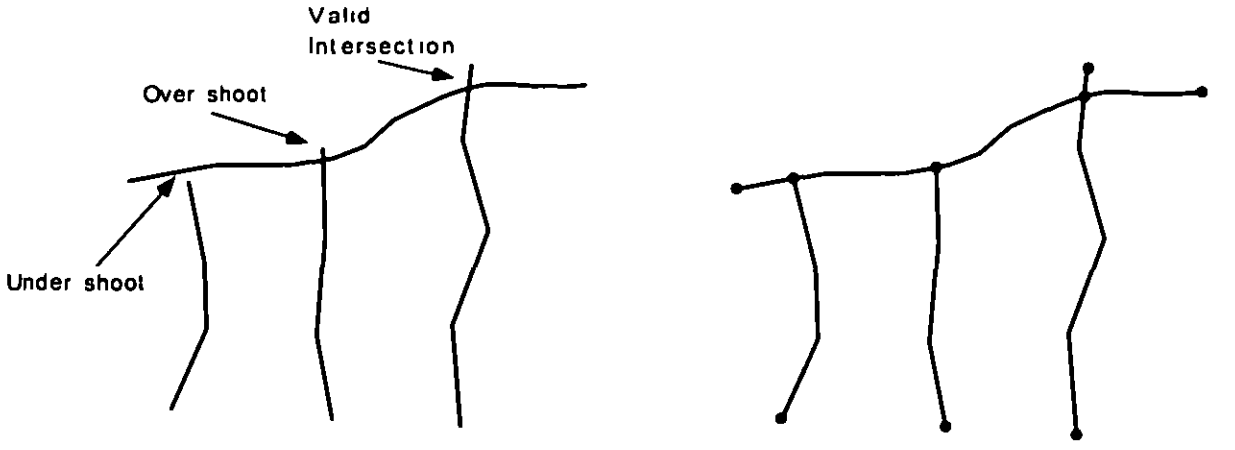


Page 28

Resolve Intra-Layer Topology

Topology is the connectedness of data that is the description of the way data is connected as opposed to the location shape or attribution of the data A common example of topology is a road network The fact that a road connects New York City and Washington DC is topology regardless of the number of curves in the road or whether the road is asphalt or concrete A topology exists in all data formats even though not explicitly stated For example the topology of a Digital Elevation Model (DEM) is a grid If there are systematic errors in the DEM data then work must be done to fix the topology of this layer of data

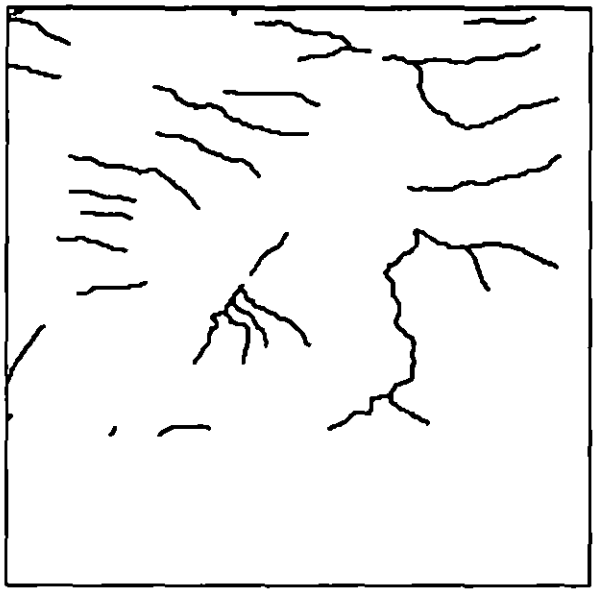
Intra-Layer Topology Example



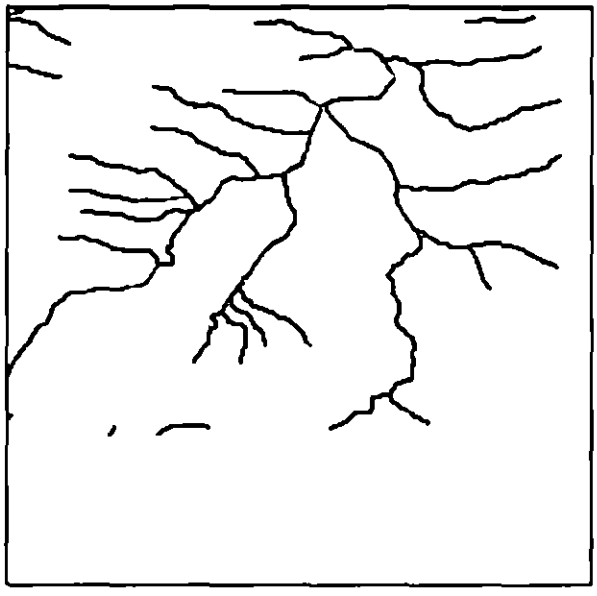
Cleaning Digitization Errors

Page 29

Intra-Layer Topology Example

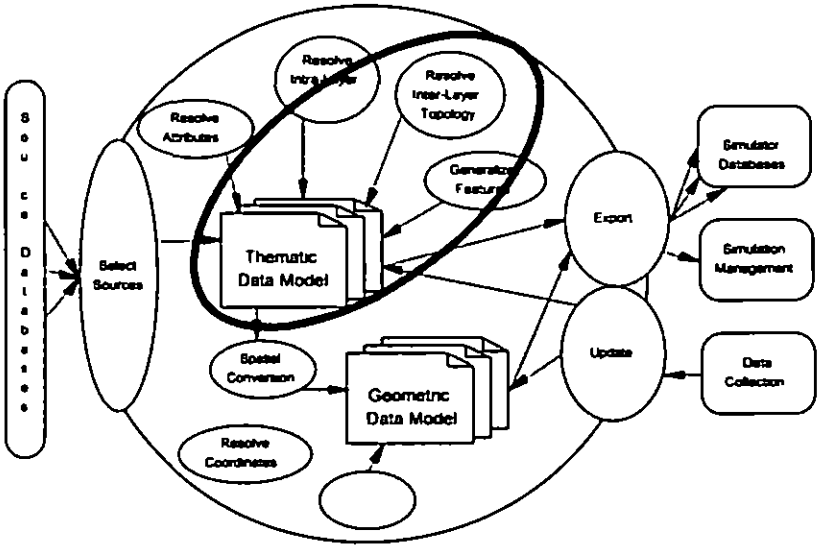


Missing river segments..



..Digitized from geo-registered imagery

Resolve Inter-Layer Topology



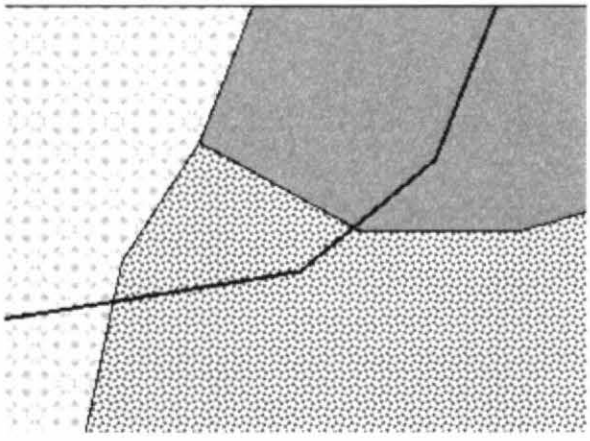
Page 31

Resolve Inter-Layer Topology

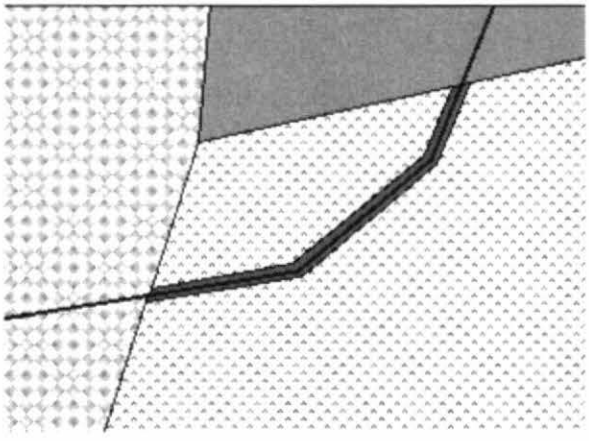
A goal of the Synthetic World is to make a consistent representation of the Real World If a physical element is represented in more than one layer then those representations must agree For example rivers may exist in a hydrography layer as an attributed line feature However the same river may exist in a Digital Elevation Model (DEM) implicitly as the channel (or valley) between two hills Now if for some reason the river line is not positionally consistent with the channel then either the river must be moved to the channel or the ground reshaped to fit the river line

Inter-layer topology may be implicitly defined by ensuring that the representations in each data layer do not contradict each other or explicitly defined through a more sophisticated data representation referred to as integrated topology Integrated topology is important when generalizing the Synthetic World to produce Level of Detail definitions and dynamic modifications to the environment

Inter-Layer Topology Example

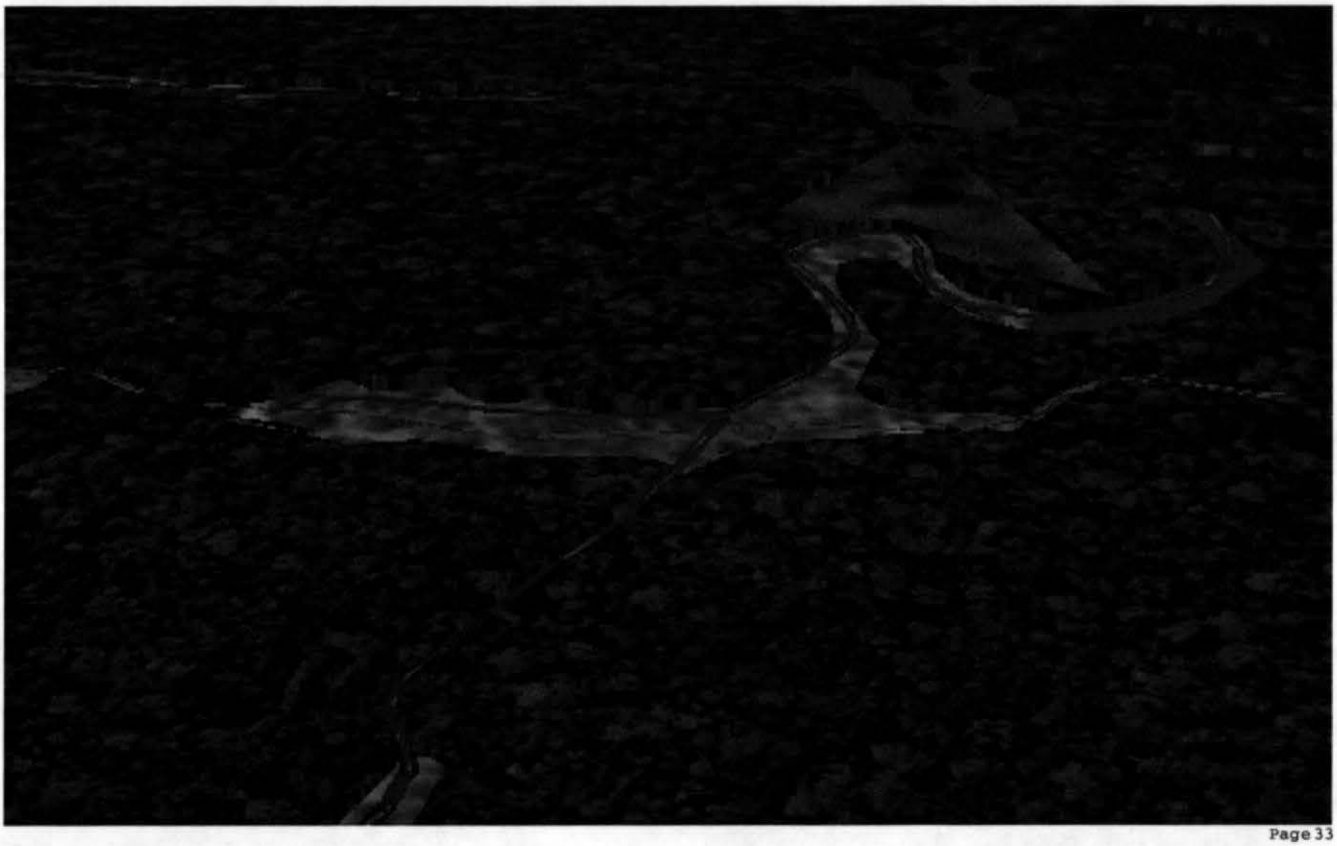


A River passing under a forest canopy



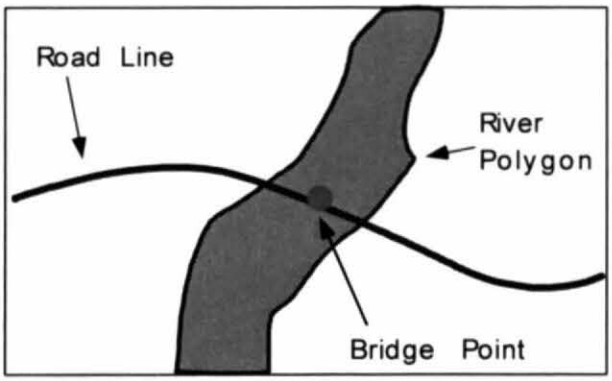
A River cutting through a forest canopy

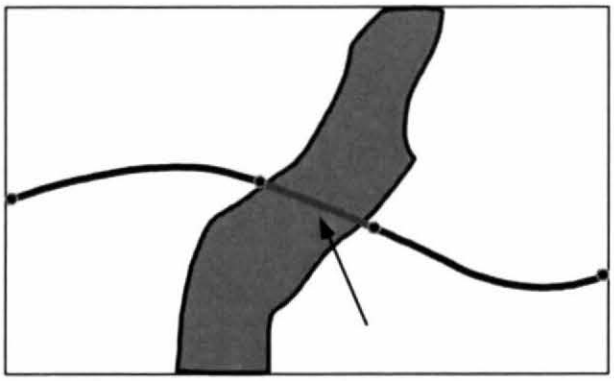
Inter Layer Topology Example



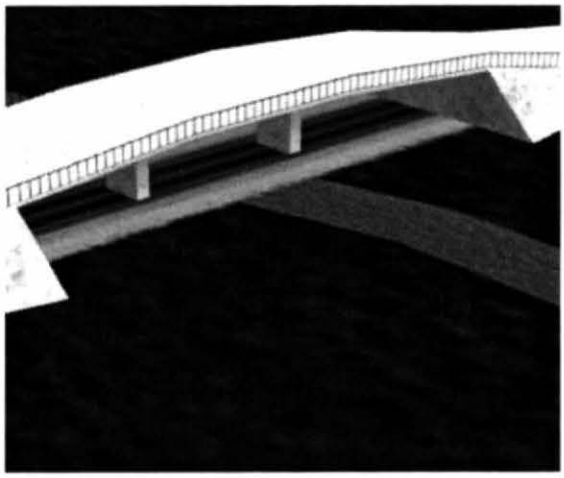
Page 33

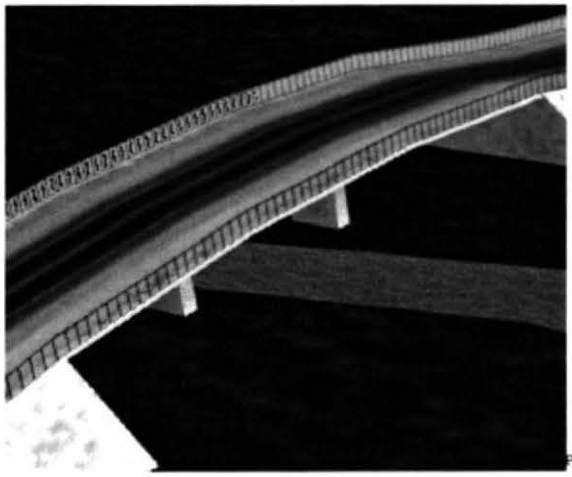
Inter Layer Topology Example





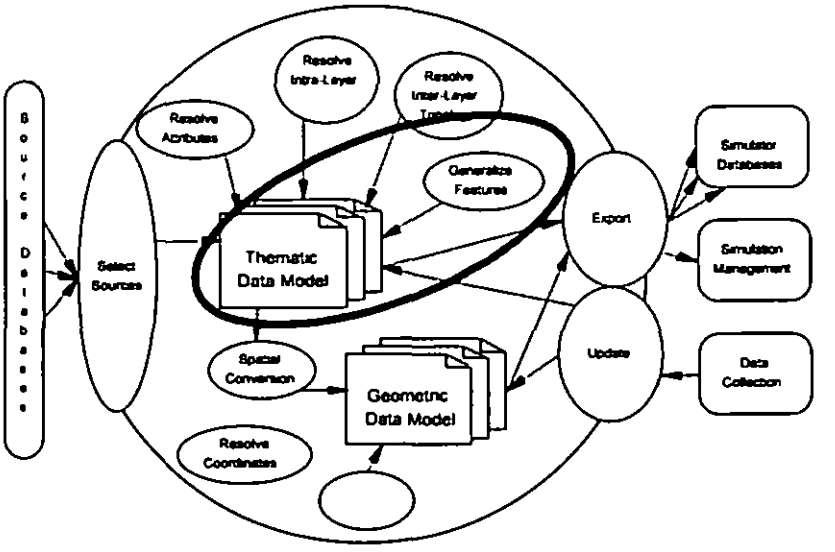
Road segment has been cut away and a bridge segment inserted.





Page 34

Generalize Features



Page 35

Generalize Features

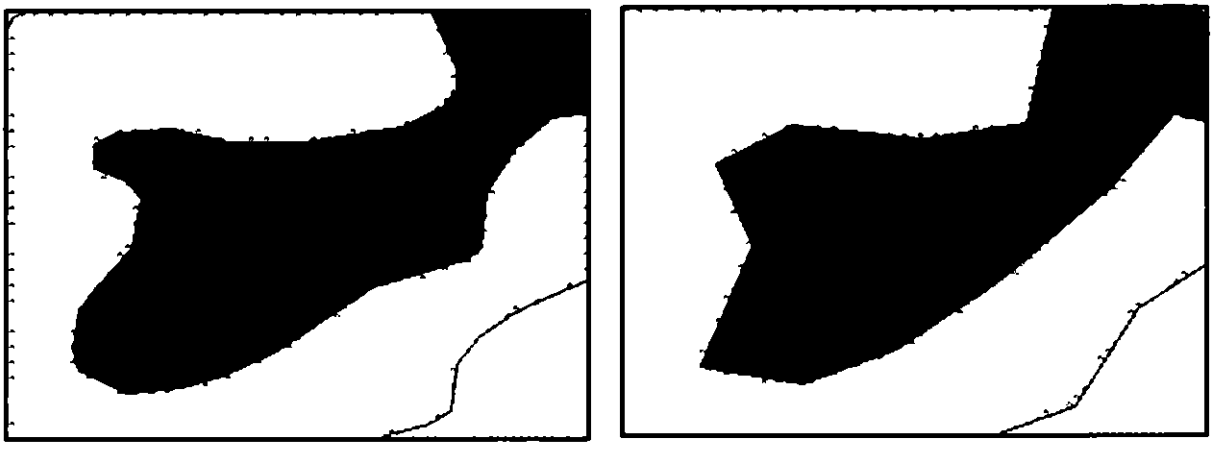
Generalizing is the process of removing complexity within the data to make versions which have less detail Two forms of generalization exist thinning and dissolving Thinning removes spatial resolution where dissolving joins features of like attributes

Let s use a linear river feature as an example of thinning If the original river bends 140 times as it flows between two cities a generalized version of the river may have only 100 bends The intra-layer topology is not changed by this operation the river still connects the same two cities The inter-layer topology however may be quite disrupted by this operation Recall that sharp bends in a river exist because of intervening terrain removal of a river segment may cause the river to flow over a hill

An example of Dissolving would be to join sandy clay and "silty clay" into clay" This operation does change the topology of the data since features are removed and replaced with simpler ones

A major problem with generalization is that the mechanisms for maintaining both the generalized and ungeneralized data as a hierarchy
cumbersome

Geometric Weeding



The geometric complexity of these areas has been reduced

Page 36

Feature Merging

Similar Features

Features with unique secondary attributes

Dissolved into single feature

Dissolved into single feature

Page 37

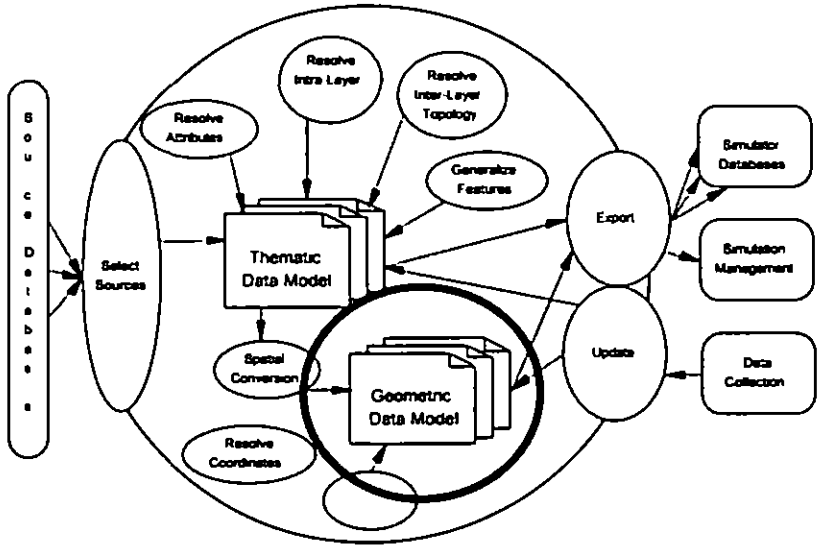
Feature De-aggregation

Area Features Built up Areas

Point Features Buildings

Page 38

Geometric Data Model



Page 39

Geometric Data Model

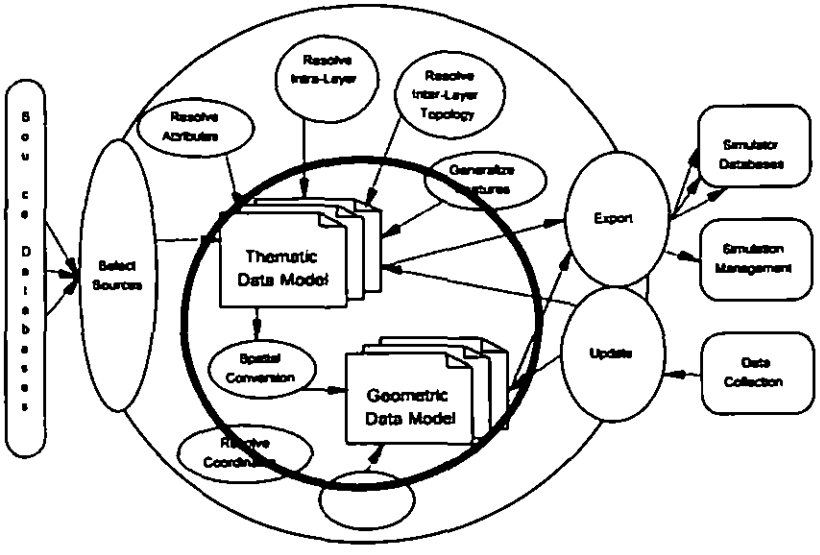
A Geometric Data Model is defined by the WRM as the integrated 3D geometric representation of both the topology and topography for a geographic area (volume) which is not optimized for a specific simulator but which can be compiled into simulator databases suitable for image generation and Computer Generated Forces (CGF) behavioral reasoning

Many data representations may be included in this model one for each level of abstraction For example a geometric model may be created for high flyers and another for ground forces As well a geometric model may be created for the visual system and a more detailed model for the radar system

The separation of the Thematic Data Model from the Geometric Data Model is, by some accounts, a conceptual flaw The ideal Synthetic World is likely to be an object oriented model of the world with both thematic and geometric components

The current WRM does however reflect the current division between types of operations provided by two classes of tools GIS and 3D modeling

Thematic to Geometric Conversion



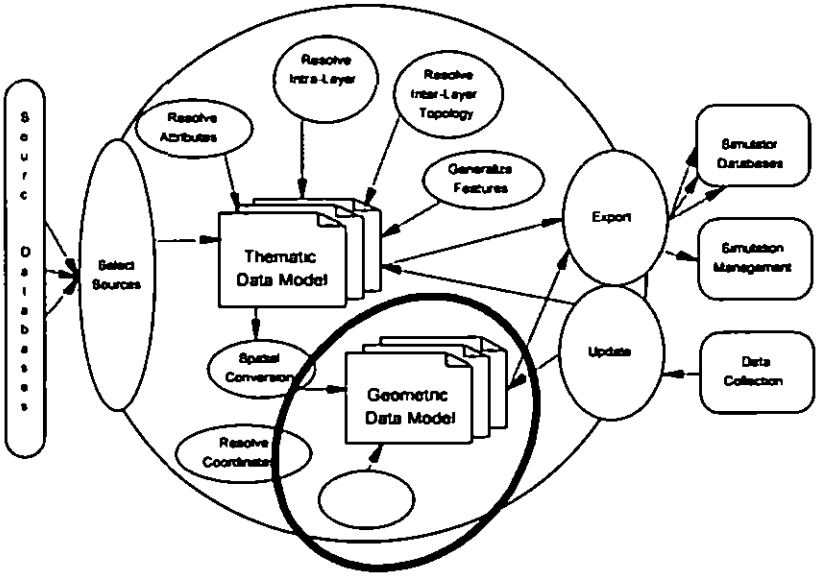
Page 40

Thematic to Geometric Conversion

The WRM calls the process of creating a Geometric Data Model from the Thematic Data Model thematic to geometric conversion The reference model does not imply how this conversion is performed Many processes in use today have plenty of tools to get the job done S1000 EaSIEST MultiGen Coryphaeus Designers WorkBench TARGET Arc/Info are but a few of the tools used Many custom-built utilities are in use as well

Regardless of the implementation the assumption is that all the necessary information to make the geometric model is included in the Thematic Data Model If human interpretation is required to "fill in the gaps then the process of 3D modeling is required

3D Modeling



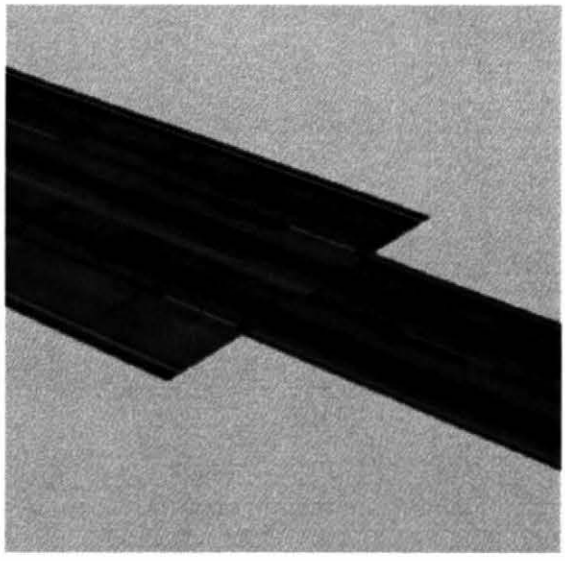
Page 41

3D Modeling

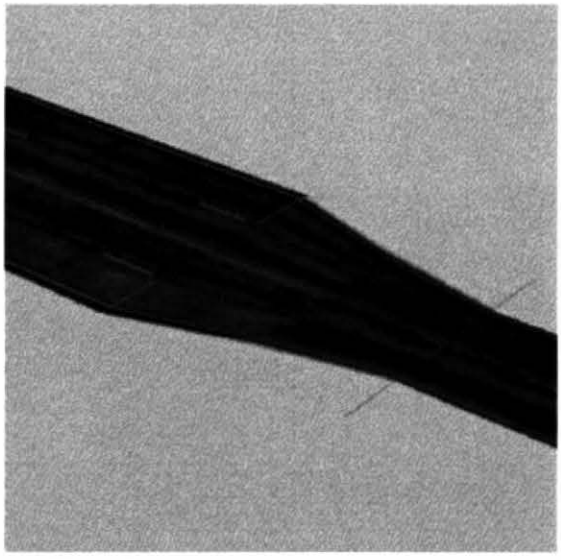
This is the process of adding detail to a geometric model based on human interpretation Many of the same tools listed above are used for 3D modeling

As long as the tools which do the thematic to geometric conversion do not make a geometric model which satisfies the aesthetic desired by the customer base 3D modeling will always be required

3D Modeling Example



As Created by Thematic to Geometric Conversion



After 3D Modeling

Page 42

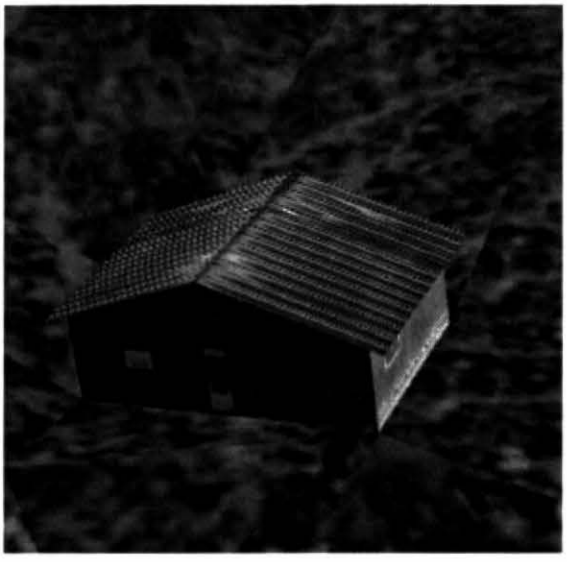
3D Modeling Example

Road intersection before...

...and After 3D Modeling

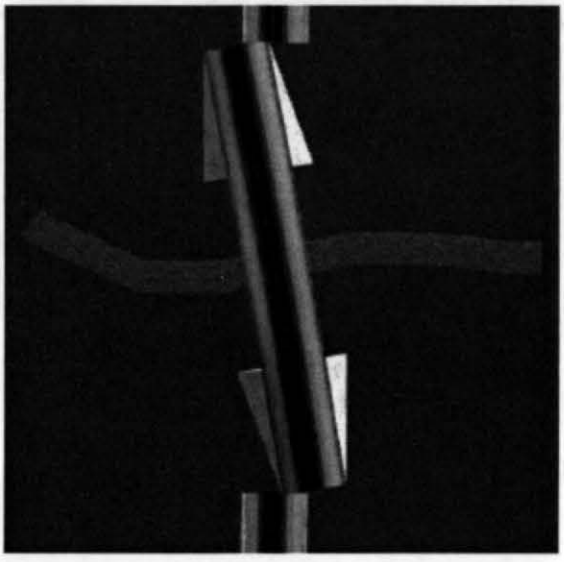
3D Modeling Example

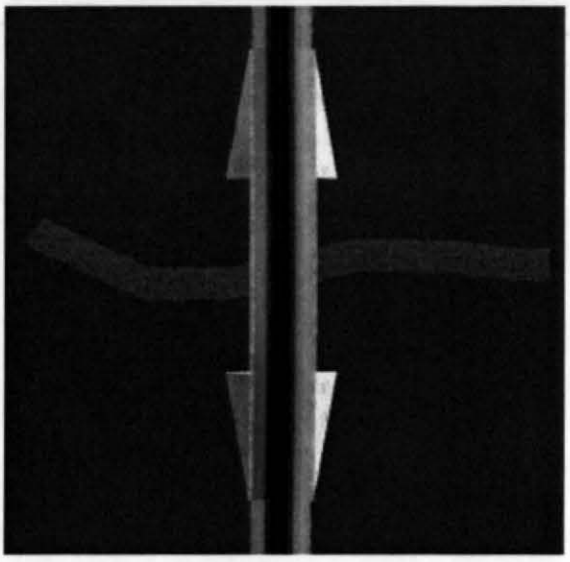
Automatically placed building which does not conform to surface



After 3D modeling of surface to conform to building

3D Modeling Example

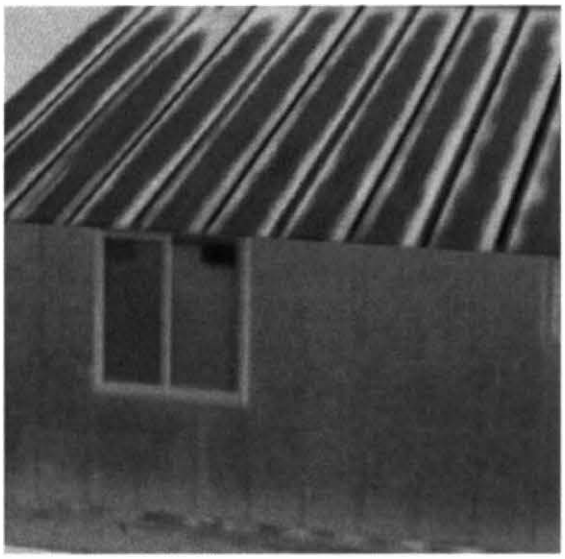




Problem caused by lack of orientation precision in DFAD

Page 45

3D Modeling Example

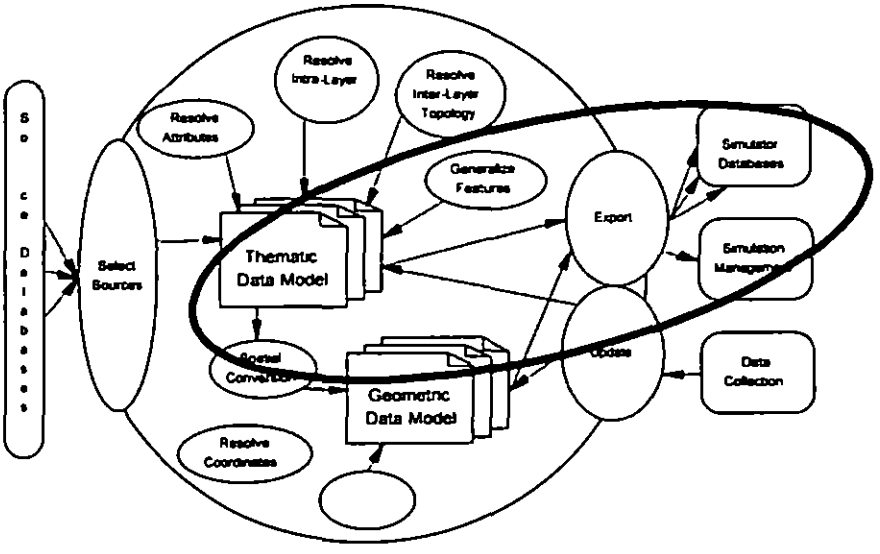


Building w/ textured wall where texture includes a window

Building w/ window modeled as hole in wall

Page 46

Export



Page 47

The Export operation is used for two different purposes

- Compilation

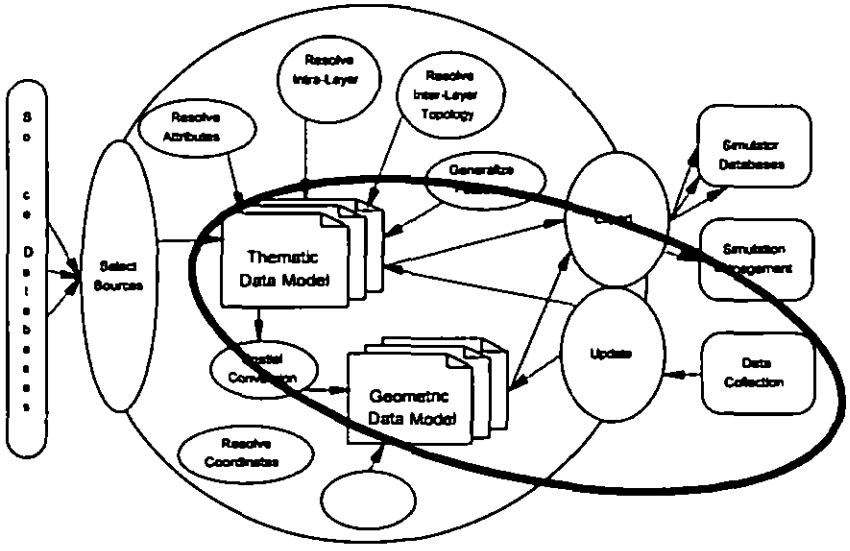
  This is the act of extracting only the data necessary for a specific task then compiling that data into a form which is optimized for that task

- Exchange

  This is the act of packaging the entire Synthetic World definition into a transmittal another user to be considered as source for their Synthetic World

  See Part 3 "Exchange the Data"

Update

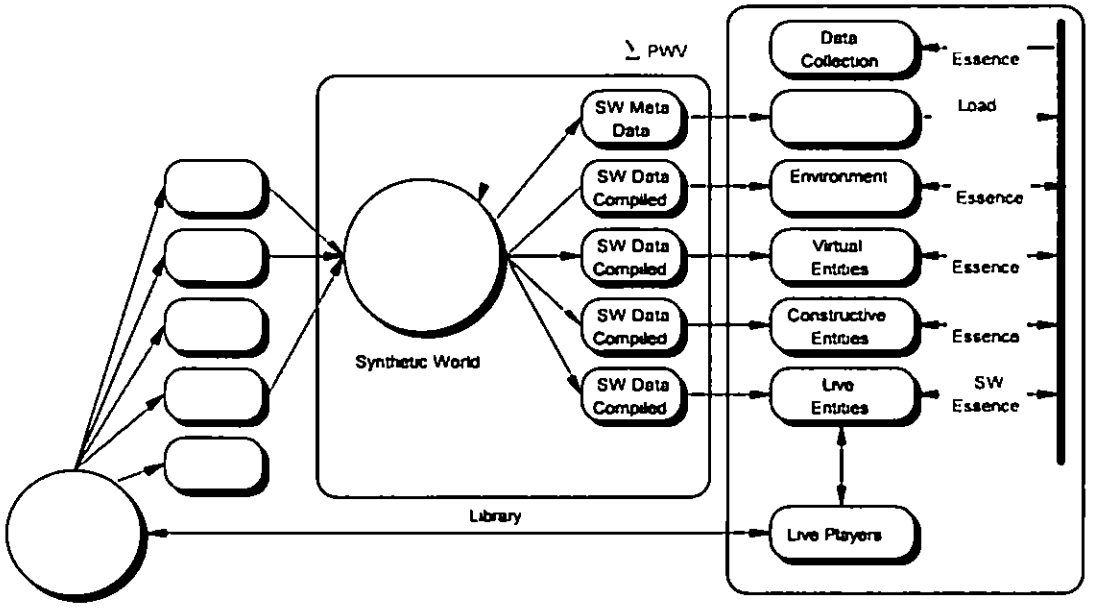


Page 48

Update

In theory updating is used to re-integrate snap shots of the Synthetic World after the world has been modified by a running simulation

Summary



Page 49

Expose the Process

-You should know how the World Reference Model models the operations which must be performed on source data to prepare it for use in distributed simulations

Communicate The Solution

-You should understand that the Environment Model, Thematic Data Model and Geometric Data Model be explicitly defined, distributed and understood

-And that a mapping between the Environment Model onto the Thematic and Geometric Data Models defines the contents of a Synthetic World

Exchange the Data

-Finally, you should be able to recognize the difference between Product Specifications and Format Specifications

Multiple Image Suppression in a Low-Cost Visual System

Ronald G Moore
Evans & Sutherland Computer Corporation
Salt Lake City, Utah
rmoore@es com

ABSTRACT

To maximize display resolution scene density and image quality at an affordable cost the U S Army Close Combat Tactical Trainer (CCTT) employs a 15Hz image update rate visual system When operating at this low image update rate visual anomalies occur which hinder the training task Multiple imaging is one of the most serious visual anomalie observed in a low update rate visual system Multiple imaging negatively affects image resolution and can cause loss of situation awareness and in some cases simulator sickness A technique called Multiple Image Suppression (MIS) is used in the Commander s Popped Hatch (CPH) panoramic display in the CCTT visual system to significantly reduce the negative effects of multiple imaging

This paper introduces the reader to the artifacts of multiple imaging that result from an image update rate that is less than the display refresh rate It describes the side effects that can occur as a result of using Multiple Image Suppression It describes the Multiple Image Suppression technique as implemented in the image generator and used on the Commander s Popped Hatch panoramic display of the M1A1 M1A2 and M2A2/M3A2 manned modules on the CCTT program This paper explores the cost and performance benefits of Multiple Image Suppression And finally the paper examines expanded uses of the Multiple Image Suppression technique

ABOUT THE AUTHOR

Ronald G Moore is a Principal Engineer and the Manager of Ground and Helicopter Systems Applications Engineering at Evans & Sutherland Computer Corporation Mr Moore has served as the Lead Visual System Project Engineer on the CCTT Program from pre proposal through present Mr Moore is a member of the Distributed Interactive Simulation Standards Steering Committee and the Chair of the Simulated Environment Working Group Mr Moore was employed by the Boeing Visual Flight Simulation Laboratory for eight years where he contributed to research in cockpit automation technology and air combat team training using distributed network technology He has been involved in image generation display and database development for over ten years He received his BSE from Brigham Young University in 1983

Multiple Image Suppression in a Low-Cost Visual System

Ronald G Moore
Evans & Sutherland Computer Corporation
Salt Lake City Utah
rmoore@es com

INTRODUCTION

With each new generation of image computing technology suppliers compete to provide the best imagery at the lowest cost The definition of best imagery is subjective but it is generally accepted to mean the highest display resolution (i e best pixel performance) the maximum scene density (i e best polygon performance) and the finest image quality (i e sub-pixel sampling, image antialiasing, texturing and shading) at a 60Hz image update rate or higher

Unfortunately not all applications can afford this definition of best imagery For each application the most cost effective use of image rendering resources is achieved at the lowest update rate acceptable to the target application This assures the maximum object density and greatest scene complexity at the lowest cost Ground combat simulators have placed a particularly high value on low cost but the inherent scene density in a ground environment requires an extremely dense scene To satisfy these conflicting objectives ground combat simulators have historically operated at image update rates between 15Hz and 30Hz. In ground warfare tactical trainers like SIMNET and CCTT (Close Combat Tactical Trainer) a 15Hz image update rate is accepted For precision gunnery training a minimum of 25 30Hz has been considered essential The U S Army PGT (Platoon Gunnery Trainer) and COFT (Conduct of Fire Trainer) systems and the AGTS (Advanced Gunnery Training System) operate at 30Hz.

Nevertheless the best imagery is obtained when the image is computed and displayed to the eye at a rate of 60Hz or higher The challenge facing suppliers today is to deliver new techniques that provide the user with the best imagery at an affordable cost

One method applied today on the CCTT program is Multiple Image Suppression (MIS) The Commander s Popped Hatch (CPH) panoramic display of the M1A1 M1A2 and M2A2/M3A2 manned modules employ s Multiple Image Suppression to reduce the artifacts of multiple imaging This post render image shifting technique provides improved image resolution when the scene is slewing across a display and reduces the likelihood of simulator sickness Multiple Image Suppression approximates the quality of a 60Hz image update rate at the cost of a 15Hz image update rate system

This improved performance is not without limitations Some visual side effects can be introduced into the image when Multiple Image Suppression is used To properly exploit the Multiple Image Suppression technique the technical trades must be understood and the technique applied under the appropriate conditions

The following paragraphs describe the visual anomalies of a 15Hz image update rate system explain the artifacts seen in multiple imaging and introduce the technique of Multiple Image Suppression The discussion continues with a description of the side effects that can occur as a result of using Multiple Image Suppression The cost and performance benefits of the apparent higher update rate system as used in the CCTT Commander s Popped Hatch panoramic display are explained The paper concludes with a discussion of expanded use of the Multiple Image Suppression technique

LOW UPDATE RATE VISUAL ANOMALIES

To understand the visual anomalies caused by low image update rate systems two terms must be understood *display refresh rate* and *image update rate*

The *display refresh rate* is the number of times per second that the rendered image is refreshed on the display device In the visual simulation community 60Hz (60 images per second) is widely accepted as the minimum display refresh rate required in the depiction of daylight imagery This is to avoid flicker

of the image which can cause eye fatigue and other negative effects The time necessary to refresh the display once is referred to as a *field* For the purpose of this discussion a 60Hz or greater display refresh rate is assumed essential

The *image update rate* is defined as the number of times per second that a new scene is computed and rendered into the frame buffer of the image rendering system The image update rate is always equal to or an integral multiple of the display refresh rate For example when the display refresh rate is 60Hz, the image update rate is either 60Hz, 30Hz, 20Hz, 15Hz, or 7 5 Hz The time required to render a single image is referred to as a *frame*

Having an image update rate lower than the display refresh rate manifests no visual anomalies when the image is static In other words when the image eyepoint is not moving (translation or rotation) and there is no movement of objects within the scene no visual anomalies will be observed However when the image update rate is lower than the display refresh rate and there is any motion in the image a few anomalies can be observed object stepping image stepping, and multiple imaging

Object Stepping

Object stepping is defined as an incremental spatial discontinuity in the continuous motion of an object moving through the rendered image Specifically, it is the discontinuity in motion caused by the object s position being computed and displayed at a rate where the discrete displacement is visible to the viewer

Erratic or spastic motion of objects within the rendered image may result from inconsistencies in image update rate or from discontinuities in eyepoint or vehicle movement Erratic motion of the image caused by systems that do not operate at constant update rates as may occur during image processing overload will not be discussed In a well-designed simulation visual system pixel and polygon processor load management techniques seldom allow overload to occur Also with the emergence of low-cost distributed network simulation, a significant amount of the erratic behaviors of the vehicles in the visual scene are due to dead reckoning and network latencies For the purpose of this discussion I will ignore inconsistency in movement due to temporal or spatial errors in the computed location of an object Smoothing of vehicle motion and time rectification can be performed to minimize the negative effects of dead reckoning and network latencies I will focus only on the anomalies of a low image update rate However these sources of erratic motion must be addressed in the final ready to-train system

When a low image update rate is used in the visual rendering system as in the 15Hz image update rate in CCTT three predominant sources of object stepping can be easily identified

One source of object stepping occurs when the eyepoint of the simulated vehicle is moved (translated) through the simulated world Objects in the rendered image appear to step as the eyepoint moves past the objects Objects near the eyepoint appear to move through the image faster than objects in the distance The relative motion of objects based on viewer movement and distance to the object is referred to as parallax A simple analog in the real world will help reinforce the concept that translation of the viewer s eyepoint produces a significant amount of relative motion within the scene As you drive down the road you can easily observe how quickly the sign posts and road signs pass by your peripheral vision while the sun or moon off in the distance seems to exhibit almost no movement at all This is object stepping caused by translational movement of the viewer s eyepoint

A second source of object stepping occurs when the ownvehicle eyepoint is stationary (no translation or rotation) in the simulated world and other simulated vehicles are moving These other vehicles may appear to step within the image depending on the distance from the viewer to the other vehicles and the rate of motion of the other vehicles More specifically the discrete displacement of a vehicle is visible to the viewer based on the distance to the vehicle and the distance moved by the vehicle in a single frame Consequently, vehicles near the viewer appear to step more than vehicles in the distance This object stepping is caused by translational movement of other vehicles

A third source of object stepping occurs due to motion in the image that is related to changes in object shape Motion such as observed in a simulated explosion may appear to grow in size in discrete steps The change in motion of the simulated explosion appears no faster than that of the image update rate This is object stepping caused by shape changes of the object.

Object stepping can be observed at all image update rates that are lower than the display refresh rate Stepping anomalies are more pronounced at the lower image update rates such as 15Hz. Stepping can be reduced at a 30Hz image update rate and practically eliminated at a 60Hz image update rate

However if the displacement of the object is extremely large in a single frame it appears to step even at a 60Hz image update rate This is because the moving object is computed at discrete intervals At near distance a fast moving object only appears at a few positions in the image as it moves through the scene This does not provide the necessary stimulus to perceive continuous motion even at a 60Hz image update rate If the object motion could be presented to the eye as a blur of the object, as would naturally be perceived the object would not appear to step

Image Stepping

Image stepping is defined as an incremental spatial discontinuity in the continuous motion of the rendered image Specifically it is the discontinuity in motion caused by the image position being computed and displayed at a rate where the discrete displacement is visible to the viewer When the ownvehicle eyepoint is at a fixed location (no translation) in the simulated world and the vehicle is rotating image stepping is observed The entire image steps to each new location based on the rate of the rotation

This type of stepping is an artifact of simulation A real vehicle moves within the real world however in the simulator the simulated world is moved around the simulated vehicle For example when the turret is commanded to turn the simulated turret does not move the imagery on the fixed location display is moved to simulate the turning of the turret The image stepping is caused by rotational movement of the viewer s eyepoint with respect to the simulated world

Image stepping is observed at all image update rates that are lower than the display refresh rate and is even observed at a 60Hz image update rate if the rotation rate is extremely high

Multiple Imaging

Multiple imaging is a physiological visual artifact which occurs when the viewer becomes fixated on a stepping object in a stationary image or on a stepping image Multiple imaging is the perception of multiple replications of the same image each slightly shifted from the other observed when viewing a moving object or image in which the update rate is less than the display refresh rate

In the case of image slewing as the image moves across the display the observer s eye intuitively tracks the objects in the image Each time the image rendering system updates the scene the new image appears at the exact position expected by the tracking eye however the intermediate refreshed images are also presented at this same position on the display Because the eye continues to track at the rate of the moving image the repeated images are focused on different areas of the retina The result is the perception of replicated images The separation between these replicated images increases as the apparent velocity of the image motion increases The number of replicated images is the ratio between the display refresh rate and the image update rate A display refresh rate of 60Hz and an image update rate of 30Hz result in double imaging a display refresh rate of 60Hz and a 15Hz image update rate result in quadruple imaging In CCTT a display refresh rate of 60Hz and an image update rate of 15Hz results in four separate replicated images

Multiple imaging is most frequently observed when the entire image is slewed across the display but can also be observed when an object is moving within the image

Slewing Images In ground based training systems multiple imaging is most noticeable during angular movement (slewing) of the simulated image This is because ground vehicles have relatively low speeds (translational velocity) compared to their rapid turning rates (rotational velocity) In fact for most ground vehicles the observed image motion is a result of slewing the turret or turning the vehicle Even an independent sight like the M1A2 CITV which is used to scan for targets generally pans from

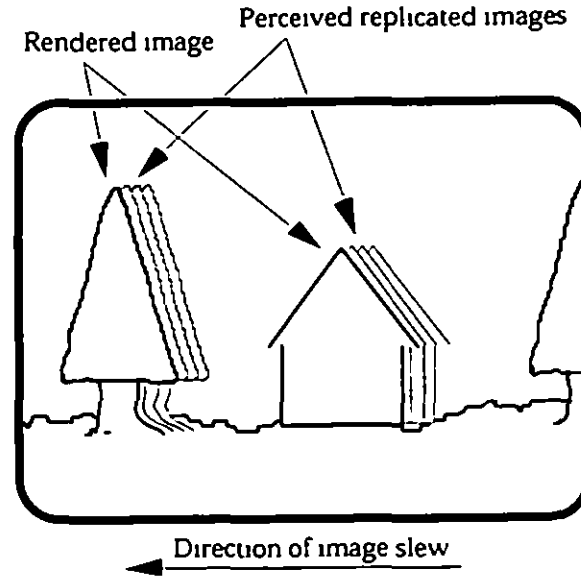


Figure 1 *A simulation system which operates with an image update rate lower than its display refresh rate*

suffers from the negative effects of multiple imaging
side to side producing rapid rotational motion Figure 1 depicts the artifact of multiple imaging in a 15Hz system caused by slewing Note that the initially rendered image is observed to be repeated three additional times at different locations to the observer s eye

Moving Objects Multiple imaging of moving objects can be observed in systems where the update rate is less than the display refresh rate whenever there is constant relative motion within the image When a vehicle moves across a stationary image and the eye tracks the vehicle the viewer may observe multiple imaging of the vehicle This type of multiple imaging will be most noticeable when the target vehicle is crossing the scene perpendicular to the line of sight This is the same effect that occurs when the image is slewed due to turret rotation The eye tracks (fixates on) the moving vehicle and since each of the three additional intermediate images are rendered in the same location on the display the eye perceives the vehicle to be four places on the retina Figure 2 depicts the multiple imaging of a moving vehicle in a stationary image with a 15Hz image update rate

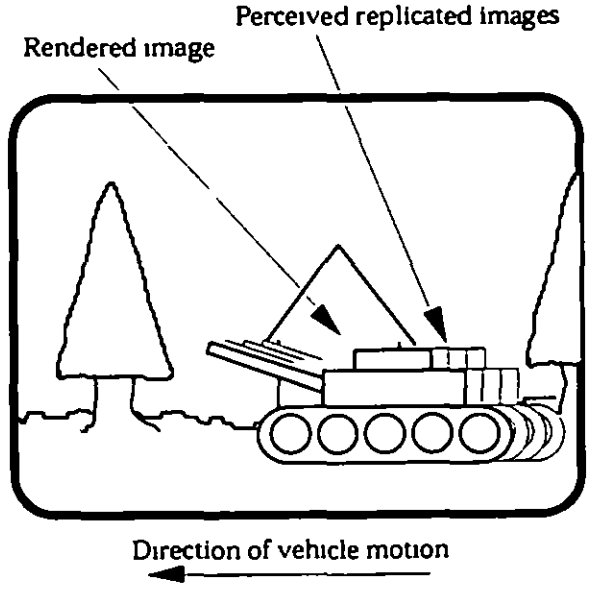


Figure 2 *Target stepping caused by target motion in a stationary image*

Negative Effects of Low Image Update Rates

Multiple imaging object stepping, and image stepping can cause both degradation in image resolution and simulator sickness

Resolution Degradation Multiple imaging results in a dramatic degradation of image resolution The higher the rate of image motion the lower the image resolution In fact, when the replicated image displacement is as little as 25% of the static image resolution the image resolution begins to perceptibly degrade This means that when the displacement of the replicated image is only a fraction of a pixel resolution is degraded This image resolution continues to degrade as the rate of image motion increases

The result of image stepping and multiple imaging due to slewing of the image is a dramatic decrease in target detection recognition and identification ranges The degraded resolution simply reduces the range in which details in the image can be resolved

The result of object stepping and multiple imaging, due to object motion in a stationary eyepoint image is an artificial increase in target detection range and a decrease in target recognition and identification ranges In a stationary image a slow moving target may be detected easily in a low update rate system because it s motion is more pronounced due to object stepping Yet, when the target is tracked it will multiple image the resolution will degrade and the viewer will struggle to recognize or identify the target

It is important to understand that this degradation in image resolution is similar for all image update rates that are less than the display refresh rate That is an image update rate of 30Hz and 15Hz will experience a similar degradation in resolution at the same angular slew rate This is because the image displacement is a function of the rotation rate of the viewing eyepoint image not the image update rate

Simulator Sickness Simulator sickness is also of major concern with low image update rate systems Object stepping, image stepping and multiple imaging are very severe in moving eyepoint systems and are potential causes of simulator sickness

Another source of simulator sickness is the disparity in sensory cues due to system latency I will not discuss this concern but it must be addressed in a low update rate system

Object stepping and multiple imaging of objects moving in a stationary eyepoint image are troublesome to target detection recognition, and identification, but are not likely to be the cause of simulator related sickness

Image stepping and multiple imaging due to slewing is very unnatural and can cause viewer disorientation confusion nausea and feelings of vertigo Concern over simulator sickness in the large panoramic 360° Commander s Popped Hatch panoramic display on the CCTT M1A1 M1A2 and M2A2/M3A2 manned modules motivated implementation of Multiple Image Suppression A significant number of viewers reported feeling dizziness and nausea as a result of the demonstrations that took place while prototyping the panoramic Commander s Popped Hatch We identified the image stepping and multiple imaging in the moving eyepoint image as the source of the dizziness and nausea

MULTIPLF IMAGF SUPPRFSSION

Multiple imaging can be avoided by assuring that the image update rate matches the display refresh rate This is the solution used in most flight simulators in service today and the solution that technology trends point to as the long term answer to the multiple imaging problem

However utilizing this solution for CCTT would require increasing the cost of the image rendering system by a factor of four or degrading the resolution and scene complexity by the same factor of four An alternative is to provide a method that can eliminate the artifacts of the low image update rate without significantly increasing the cost of the image rendering system

Multiple Image Suppression is a technique designed to reduce the artifacts of multiple imaging caused by image slewing with minimal impact to processing performance Essentially the image rendering system over computes a slightly enlarged horizontal field of view image each processing frame Figure 3 depicts this over computed image When image slewing is introduced into the computed scene in response to rotational inputs from the host computer the direction and magnitude of the motion is computed This motion information is processed in the image rendering system using an interpolation algorithm to determine the incremental change required in the image for each of the intermediate refresh fields during the update frame The displayed image is shifted appropriately within the over-computed image to accommodate the change in viewing direction in each display refresh frame

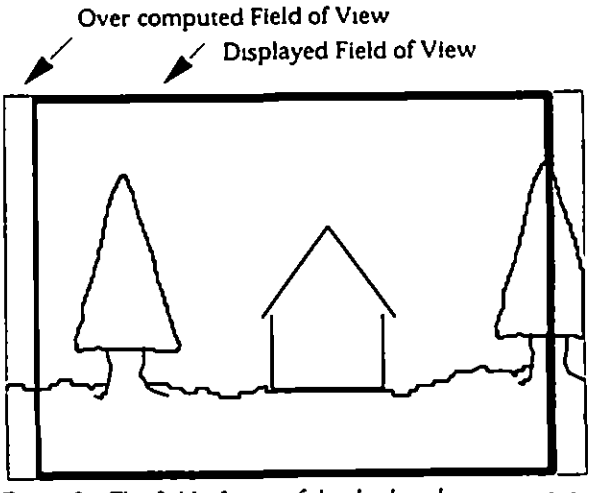


Figure 3 *The field of view of the displayed image is over computed such that the image can be shifted to provide correct placement of the intermediate images between image updates*

The computed field of view must be large enough to cover the field of view currently being displayed and also the surrounding field of view which could be visible before the next image update The size of the over computed field of view is a product of the maximum slew rate for which multiple imaging must compensate The size of the additional field of view defines the performance impact of implementing Multiple Image Suppression

The information necessary to shift the image is sent directly to the frame buffer so that placement of the current image and the next image is exactly known Interpolation in image position provides placement of the intermediate images This assures that the intermediate images are in the correct location as if they were computed at the display refresh rate This interpolation (rather than extrapolation) also allows for compensation of acceleration in image slewing and provides correct control of the intermediate images even in polygon or pixel processor overload conditions Interpolation is possible because the command to control image shifting is sent to the frame buffer in approximately one frame while the data to render the image takes approximately two frames The location of the next frame is known before the previous frame is rendered The interpolation algorithm correctly computes the intermediate image position at all times failing only under the most extreme polygon or pixel processor overload conditions

MULTIPLF IMAGF SUPPRESSION TECHNOLOGY TRADF

Multiple Image Suppression is not without side effects Understanding the limitations of Multiple Image Suppression was key to its successful application on the CCTT program

Commander's Popped Hatch Display Analysis

As noted earlier (and prior to incorporation of Multiple Image Suppression) a significant number of viewers reported feeling dizziness and nausea as a result of demonstrations in the prototype Commander s Popped Hatch panoramic display This raised concern that a 15Hz update rate would be unacceptable to the CCTT users A detailed pre contact analysis of all of the CCTT viewing devices revealed a number of key observations

First, when viewing from the design eyepoint of the Commander s Popped Hatch display the only physical spatial reference for the viewer comes from the image presented on the ring of surrounding monitors This means that all spatial references including peripheral vision are provided by the computed imagery Consequently when the imagery is moving, and multiple imaging occurs a stable spatial reference is lost This was the source of the simulator sickness

Second multiple imaging does not induce dizziness and nausea in the prototype vision blocks that were operating at the same 15Hz image update rate as the Commander s Popped Hatch This is because only a limited amount of the viewer s field of view is occupied by the computed imagery The spatial references are provided by the mechanical structure of the vision blocks and surrounding fixtures

Third when the image is slewing, the loss in resolution is observed in all of the display devices as a result of multiple imaging This was the primary source of the loss in target detection identification and recognition range

Fourth the loss in dynamic resolution is less dramatic in the magnified sights than expected There are two reasons that appear to contribute to this effect limited slewing rate and object of focus It was observed that the operator did not slew the magnified sight faster than is possible to assure a focused image That is the operator limited perhaps artificially the slew rate to avoid multiple imaging It was also noted that the magnified sight slewed side to side across the area of regard while searching for targets Once a target is acquired side to side slewing stops and the sight is used to track the target More importantly the object of focus changed from the background to the tracked target While tracking the target, the target is held centered in the sight, and the multiple imaging of the background objects is no longer noticed

Fifth multiple imaging was more noticeable in built up areas of the visual database This was due to the many sharp edges particularly in the buildings Forest and grassland areas were less distracting when multiple imaging occurred

Sixth the higher the resolution of the computed image the more noticeable the loss of resolution at smaller rotational and translational velocities This is due to the relationship of the static resolution to the degradation from the displacement

With a reasonable understanding of the cause of the simulator sickness we determined that it was vital to suppress multiple imaging in the Commander s Popped Hatch panoramic display Obviously we could not artificially limit the slewing rate of the turret or the turning rate of the vehicle and the cost of adding image computational resources to allow a higher image update rate was prohibitive However because the primary motion of the ground vehicle viewing devices is slew the implementation of the Multiple Image Suppression appeared the ideal solution Multiple Image Suppression is used in the Commander s Popped Hatch panoramic display to keep costs down, while eliminating the negative artifacts of multiple imaging However like a doctor s prescription the cure may have some side effects

Multiple Image Suppression Side Fffects

We observe two side effects in the CCTT Commander s Popped Hatch implementation of Multiple Image Suppression jitter of the ownvehicle model in a slewed image and jitter of moving objects in a slewed image

Ownvehicle Model Objects in the image which are fixed with respect to the display inherit the compensation movement removed for the background scene The ownvehicle model used to provide the illusion of viewing from atop the vehicle is one such object. The ownvehicle model is in a fixed position with respect to the display device Hence when the image is moving and Multiple Image Suppression is employed the background scene is stable but the ownvehicle image jitters I use the term jitter to denote that the observed motion is a back and forth motion almost like a vibration This side effect is

very distracting when viewed for the first time but when considered in the context of how we use the Commander s panoramic view it appears to be an acceptable trade

The Popped Hatch panoramic display provides the Commander with a comprehensive view of the area surrounding his vehicle From the Popped Hatch the Commander can survey the landscape and search for targets The landscape and targets are much more important than the foreground ownvehicle model Initial results indicate that the jitter in the ownvehicle is noticed only in the first few moments of the simulation Within seconds the primary focus of the training takes over and the ownvehicle jitter is never considered again

Moving Targets Objects in the scene that are moving perpendicular to the line of sight inherit the compensation movement removed from the background scene when the image is slewed at or near the translational rate of the target object This side effect observed in the Commander s Popped Hatch is actually due to the actions of the gunner or coincidence between vehicle motion and image slewing

To understand this side effect, we need to look at the visual anomalies apparent when tracking a target in a low image update rate visual system When a vehicle is moving across the scene perpendicular to the line of sight and the image is stationary the vehicle exhibits multiple imaging as described earlier in this paper and illustrated in Figure 2

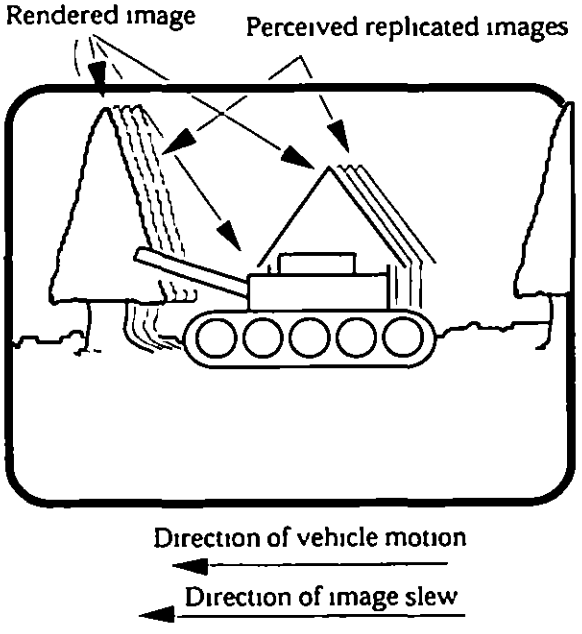


Figure 4 *Background stepping when a moving target is tracked without Multiple Image Suppression*

However in a magnified sight the vehicle is not ordinarily allowed to simply move across the scene but in fact the magnified sight is slewed to track the moving vehicle The object of interest in this case a vehicle is maintained in the center of the image Without Multiple Image Suppression the target is appear clear and crisp in the center of the display and the background exhibits multiple imaging This artifact is illustrated in Figure 4 Note that the moving target is held relatively motionless in the center of the image while the background is moving behind the vehicle

When Multiple Image Suppression is active in the Popped Hatch panoramic displays and this condition occurs the moving vehicle target inherits the compensation movement removed from the background scene This condition is illustrated in Figure 5 Note that the tank is shifted forward in the direction of the image slew The magnitude of the jitter is a function of the target s speed and range to the observer

With the low speeds of ground vehicles and the typically long engagement ranges this inherited stepping is seldom large enough to be distracting In the Commander s Popped Hatch display this is a very low probability of occurrence artifact, and consequently it is accepted as a good trade for improved target detection recognition and identification in the typical operation

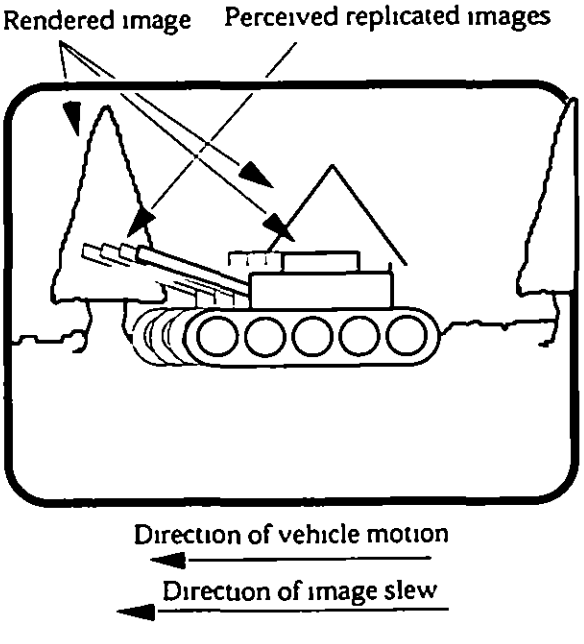


Figure 5 *Target stepping caused by Multiple Image Suppression when the image is slewed at the rate of the target movement*

The use of Multiple Image Suppression in the Commander s Popped Hatch panoramic display makes a very cost effective trade for CCTT At the cost of a 15Hz system we approach the image quality and resolution of a 60Hz system and reduce the possibility of simulator sickness

OTHER APPLICATIONS OF MULTIPLE IMAGE SUPPRESSION

To date Multiple Image Suppression has been employed only in the Commander s Popped Hatch panoramic display of the manned modules of the CCTT systems However other applications seem inevitable as improved performance is demanded at lower costs Multiple Image Suppression can be used to improve the performance of the magnified sight and vision blocks

Vision Blocks

By providing the additional polygon and pixel processing resources needed to over compute the image each vision block could utilize Multiple Image Suppression with fundamentally the same set of trades found in the Popped Hatch panoramic display system This would improve the resolution of the vision block when the vehicle imagery is slewing

Magnified Sights

The performance can be similarly improved for the magnified sights In addition to compensating for slewing Multiple Image Suppression is also capable of supporting pitching movement of the image Compensating for multiple imaging in both slew and pitch has been shown to significantly improve the ability of the gunner to resolve detail in the scene while panning side to-side and pitching up and down

However to add Multiple Image Suppression to the magnified sights a few more technical trades must be understood

In the CCTT modules the reticles are rendered in the image along with the background image To add Multiple Image Suppression to the magnified sight the symbology can no longer be rendered within the background imagery Symbology like the ownvehicle mentioned earlier is static with respect to the display The sight symbology can be provided using a post render post image shifting image mixing process This will totally avoid the symbology interacting with the Multiple Imaging Suppression This enhancement is being evaluated for insertion into the CCTT magnified sight

As noted in the discussion regarding the side effect of Multiple Image Suppression and illustrated in Figure 5 when a moving target is tracked in a magnified sight, Multiple Image Suppression can degrade the resolution of the target Jittering in the target reduces the target s recognition and identification range At typical engagement ranges and typical speed targets and with current technology magnified sights this jitter will not be observed However to eliminate this negative side effect the Multiple Image Suppression can be disabled when a target is being tracked Techniques to accomplish the activation and deactivation of Multiple Image Suppression based on natural operator actions are under evaluation

CONCLUSION

Suppliers continue to develop faster and less costly image rendering systems and users continue to demand higher display resolution finer image quality and increased object density all at the highest image update rate possible But until technology can affordably recreate the real world at 60Hz, techniques like Multiple Image Suppression must be employed to get increased performance for less money

REFERENCES

Peli E Vision Models for Target Detection and Recognition" *World Scientific Publishing Co Pte Ltd* 1995

Statler I "Characteristics of Flight Simulator Visual Systems NASA Technical Memorandum 81278 1981

ACKNOWLEDGMENTS

Special thanks to Larry Rodd for his encouragement and to Don Seymour for his graphics

Non-Linear Distortion Correction

Stephen H Gersuk (shg@ParadigmSim com)
Paradigm Simulation, Inc

1 INTRODUCTION

Distortion correction generally describes any of several meth ods to present a geometrically correct image to the eye under circumstances where the projection geometry or the projection optics introduce errors in the shape of the image Geometri cally correct means that scene elements are in their perspec tive correct positions from the eye

I inear distortions are those that can be corrected by setting an asymmetric frustum Non linear distortions are higher order distortions resulting from projection onto a curved surface or projection using a distorting lens or a combination of both

Static distortion correction describes systems in which the distortion is invariant with time because the relationship be tween eye projector and screen is fixed -- or variations are sufficiently small that they can be ignored Dynamic distortion correction accommodates cases where the distortion must be recomputed at some regular or irregular interval

There are two basic methods for correcting distortion by shaping the raster in the projector or by pre distorting the image in the image generator

Distortion correction by raster shaping relies on algebraic sig-nals derived from the sawtooth ramps that drive the horizontal and vertical sweeps Controllable amounts of each component are either summed with the ramps driving the main deflection coils or used to drive an auxiliary deflection coil to reshape the nominally rectangular raster In analog projectors these signals are created by analog integration of the ramps the current generation of digital projectors derive the ramps and correction signals digitally This method applies primarily to CRT projectors I CD projectors have a fixed raster imposed by the I CD structure although some I CD projectors simulate these raster shaping adjustments The GE I alaria series projector the only high brightness projector available for many years for simulators have an immutable raster structure imposed by an electron beam creating waves on an oil film

Distortion correction by raster shaping is typically limited in the amount of distortion it can produce because large distor tions require high currents at high bandwidths through the deflection coils Some display system vendors (notably SFOS in the UK) modify commercial CRT projectors to achieve greater range Even so I CD projectors are enjoying a wave of popularity in replacing older Talarias in high-end display systems and typically depend on distortion correction in the image generator

Non linear distortion correction (NI DC) for simulation dis play systems commonly appears in dome display systems NLDC has been a feature for high-end image generation sys-tems manufactured by Evans and Sutherland (who call it Non I inear Image Mapping) and Lockheed Martin Information Systems nee GE (who call it Comprehensive Distortion Correction) for over a decade and has commonly been implemented in software supplemented by special purpose hardware I he are generally two methods in use

The first method subdivides polygons into smaller polygons the larger the angle subtended the more extensive the subdivision These smaller polygons are then projected onto a polyhedron that approximates the surface of the dome This method is appropriate for systems that can accommodate the large resultant increase in transformations

The second method renders all polygons full size in normal perspective and then warps the image to fit the dome

Paradigm has implemented the latter method entirely in soft ware in our Vega product line for visual simulation on Silicon Graphics and PC workstations The algorithms underlying the design are presented in this paper

2 TARGET SYSTEMS

High end virtual simulators most commonly used for flight simulation employ a variety of complex and elegant display systems to provide pilots with panoramic fields of view Some of these systems are implemented as arrays of projectors in one or more tiers to provide a high resolution mosaic Others use only a pair of projectors to provide total spherical cover age for a low resolution background supplemented with high resolution area-of interest (AOI) insets The background image is itself a mosaic of channels that covers its allocated field of view The AOI may be fixed or slaved to the position of the eye head or a target Most projectors use highly distorting lenses

The rationale for the development of these expensive systems is an interesting topic in itself The Proceedings of the I/ITSEC (nee IITESC nee ITEC) and IMAGE Conferences provide descriptions of many of these systems

Some of these systems are described below

Hammer-Aitoff Plots[1]

But first a quick note about Hammer Aitoff plots or just plain Hammer plots These are the plots are commonly used to show the field of view of panoramic display systems because they show total spherical coverage in a reasonably intuitive manner The same plot can also show obstructions to the field of view (like canopy bows in a cockpit, or the nose of an aircraft or the bow of a ship) A Hammer plot is sometimes surprising it clearly illustrates for example that the sides of a rectangular channel are lower in elevation than

[1] Description and transformation from *An Album of Map Projections* US Geological Survey Professional Paper 1453 US Government Printing Office 1989

the center as shown in the first plot for SOPS It s obvious why when you think about it less obvious if you never have

Hammer plots are equal area, meaning that areas of all regions are shown in the same proportion to their true areas

Hammer plots are frequently (and incorrectly) called Aitoff plots An Aitoff plot appears very similar but is not an equal area projection

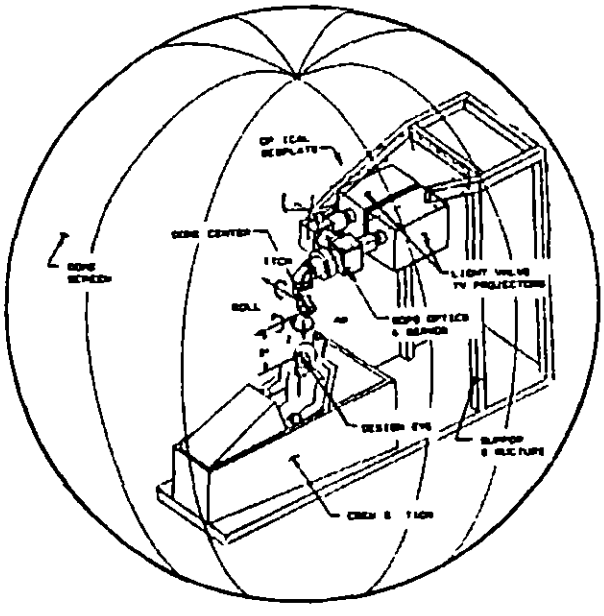
The central meridian is a straight line half the length of the Equator Other meridians and parallels are complex curves the border is elliptical The length of the central meridian is $2\sqrt{2}\ radius / scale$

The transformation from *longitude* λ and *latitude* φ on a unit radius sphere to Hammer $(x\ y)$ coordinates is

$$\text{Hammer}(\lambda\ \varphi) = \sqrt{\frac{2}{1+\cos\varphi\cos(\lambda/2)}}\begin{pmatrix}2\cos\varphi\sin(\lambda/2)\\ \sin\varphi\end{pmatrix}^T$$

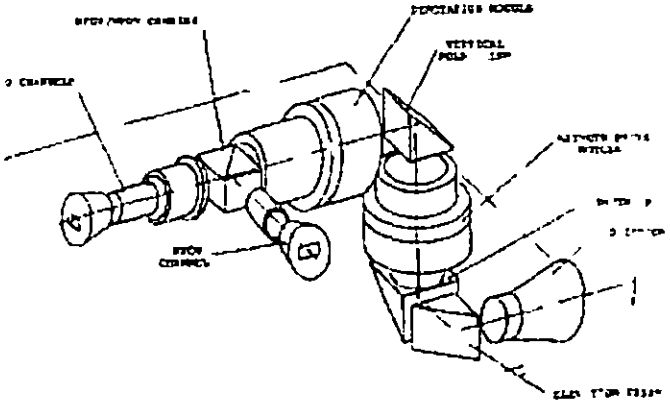
SOPS[2]

SOPS (Servo Optical Projection System) with the general layout shown below was developed for Hughes Helicopter (subsequently McDonnell Douglas Helicopter Company and now part of Boeing) by Pacific Optical



tracker and servo system slaves the projector to the line of sight ±45° elevation traversal and ±90° azimuth traversal produce a nearly unlimited field of regard The cost and complexity of the servoed optics reaps a two-fold reward first, in the reduction of the number of image generator channels required and second because only a relatively small portion of the dome is instantaneously illuminated cross screen reflections are reduced improving contrast The latte effect should not be underestimated – it s difficult to ge contrast ratios higher than 8 1 in a dome

The lens includes a beamsplitter prism with a mask tha optically combines the background and inset images As a result the background image need not be notched out for the inset



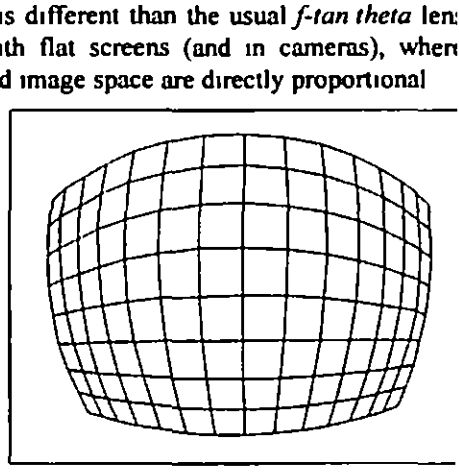
The azimuth axis for the projection optic is on the dome verti cal axis The final projection lens that forms the exit pupi swings on an arm off this axis and is displaced to one side when pointed straight ahead (SOPS 5 the most recen design has the projector s azimuth axis tilted back 15° from vertical to reduce self shadowing and increase vertical field o view)

An artifact of slewing a projector in azimuth is that is rotates the image about the projection axis This can be corrected either optically using a derotating prism or by counter rotating the channel in the IG For SOPS this function is performed in the optical train s derotation module

The SOPS design eyepoint is on the dome vertical axis 24 inches below center Head tracking in combination with an algorithm in the host, is used to control the projector pointing direction in azimuth and elevation and allows perspective correction for horizontal and vertical head motion

The SOPS lens is nominally *f theta* mapped meaning tha height in object space is directly proportional to the *angle* in image space This is different than the usual *f-tan theta* lens commonly used with flat screens (and in cameras), where heights in object and image space are directly proportional

An example of the distortion required in the object plane of the SOPS background projector is shown in the figure at right

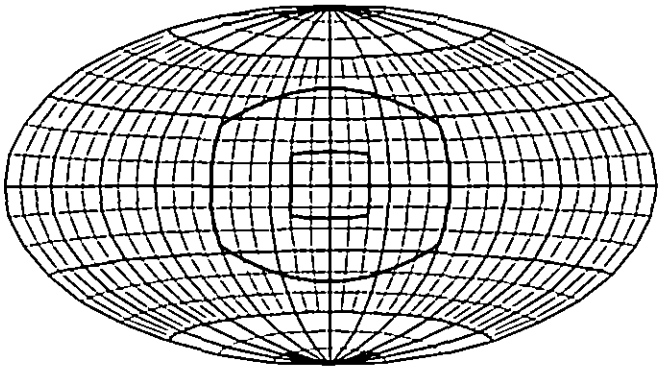


SOPS is well described in two articles in the

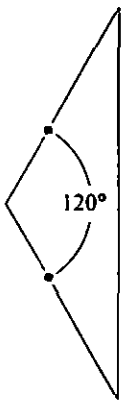
[2] Illustrations and information courtesy of Boeing/MDHC

Proceedings of the 1987 IMAGE IV Conference [3]

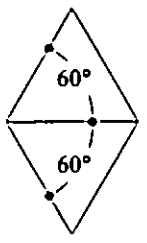
The Hammer plot below shows the instantaneous field of view for SOPS at the design eyepoint and forward line of sight



Clipping to the six planes of the view frustum for such a wide channel results in very deep corners much further from the eyepoint than along the line of sight as shown near right This consumes a lot of system capacity in drawing unneeded polygons A two channel mosaic background shown far right, provides a more efficient ground footprint

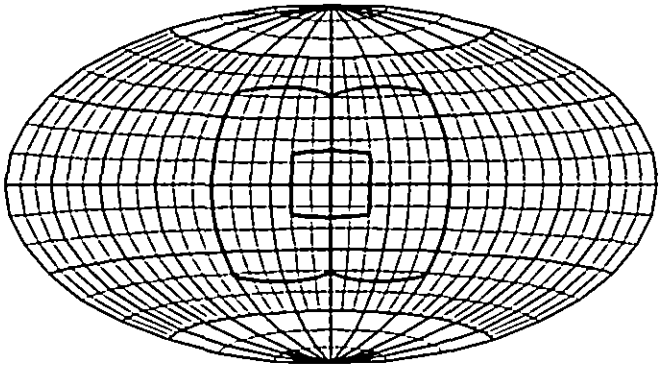


(a) Single-channel background area=1 73



(b) Two-channel background area = 1 15

Here s the Hammer plot for the two-channel background



Hemi-HIROPS

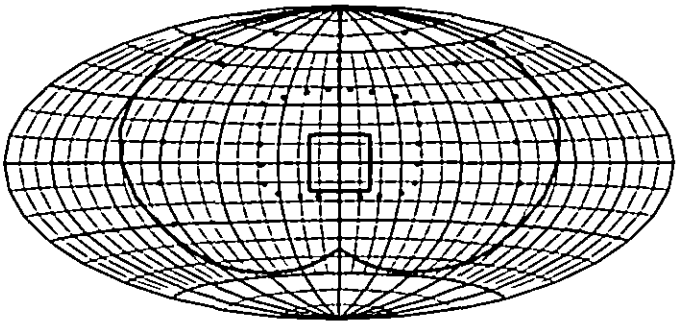
Hemi HIROPS (Hemispherical High Resolution Optical Projection System) is a dome display system developed by McDonnell Douglas Training Systems (now also part of Boeing) Many variants of the system have been built

[3] The IMAGE Society Inc Tempe AZ
http //www public asu edu/~image/Image html

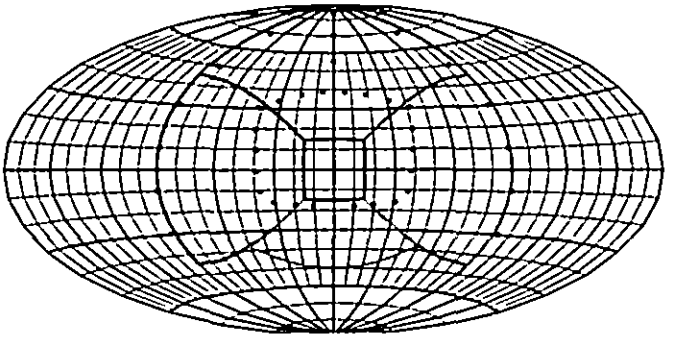
The general configuration consists of three statically mounted projectors outside the dome with imagery projected through small holes The technique is appropriately called hidden hole projection because the holes are virtually invisible from the design eyepoint

One projector illuminates the entire forward hemisphere a second illuminates the entire aft hemisphere and a third provides a high-resolution fixed inset in the forward field of view Edges between hemispheres are blended in some systems and butted in others

One example is located at the Navy s Manned Flight Simulator facility at the Naval Air Warfare Center at Pax River The fixed base 40 foot dome display was installed by Hughes (now part of Raytheon) in 1988 The display uses a two channel hemisphere mosaic with the configuration shown below Imagery from the two large channels does not entirely fill the forward hemisphere but the lower area is occulted from the pilot s view by the cockpit and fuselage The projector originally a Talaria light valve with hidden hole optics has been replaced with a Barco 1 CD light valve and Barco s standard dome lens The original 60 x 80° AOI channel has been replaced with a 30 x 30° image from a CRT projector corrected by raster shaping



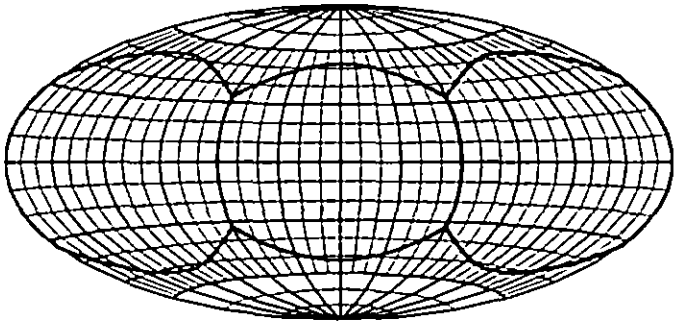
When an Onyx2 workstation is substituted for the original Compuscene IV the two-channel hemisphere mosaic is replaced with a four channel mosaic that includes a hole for the inset as shown below This provides sharper resolution and better use of polygons in the background mosaic while simultaneously reducing the total rendering time



Four-Channel Dome

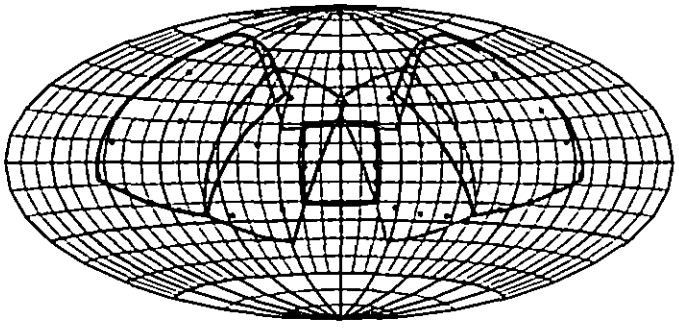
Both Sogitec and Hughes (now Raytheon) have delivered the four-channel dome configuration shown below The three

horizontal channels each illuminate a 120° x 90° field of view to provide total azimuthal coverage On some systems the original Talarias have been replaced with LCD light valves coupled via special lenses to the existing hidden hole optics A fourth projector located at the foot of the cockpit at the front of the dome illuminates the remaining area at the top of the dome



Six-Channel Dome

E&S has delivered a six projector dome with a high resolution forward hemisphere for fixed wing flight simulation The total field of view is approximately 220° x 120° and channels are edge blended to produce a continuous image



3 METHOD

General Approach

Each perspective channel s unit-distance image plane is divided into a grid Each vertex in the grid has a channel coordinate (x 1 z) and a corresponding texture coordinate (u v) Each point in the grid is projected from the eyepoint onto the dome surface Then each point is projected from the dome toward the center of the projector exit pupil onto the unit distance image plane for the projector From there a bi linear interpolation maps the projector image plane through the lens onto the projector s object plane The image plane positions thus computed are a 2 D mesh of triangles that are later rendered in an orthographic projection

In the subsequent Draw process each channel is rendered in normal perspective fashion, and then copied from frame buffer to texture memory Then meshes computed in the previous App process are rendered with orthographic projections and textured using the original (u v) coordinates When thi image is projected onto the dome a perspective-correct dis play is presented to the eye

Details of the ray trace algorithms are described below

Coordinate Systems, Data, Units, and Functions

As is usual in graphics applications a variety of coordinat systems are used convenient to the task at hand We us right handed coordinate systems oriented $+x$ right $+$ forward $+z$ up Rotations are *heading* $+y$ into $-x$ *pitch* $+$ into $+z$ and *roll* $+x$ into $-z$

a [*az el*] dome angular coordinate system *az* is the angle horizontal angle measured with a theo dolite (i e $+y$ into $+x$ opposite of *heading*) and *el* is the vertical angle measured up from horizontal (the same as *pitch*)

c [x y z] channel coordinate system origin at the channel eyepoint, axes aligned to the channel orientation

d [x y z] dome coordinate system origin is at the cen ter of a spherical or ellipsoidal dome or at the apex of a conical dome or on the vertical axis at arbitrary height for a cylindrical dome or at the design eyepoint for a flat screen

e [x y z] eyepoint coordinate system origin at eyepoint for a given channel axes parallel to dome coordinate system

o [x y z] observer coordinate system origin at the channel eyepoint position in the dome axes aligned to the observer orientation

p [x y z] projector coordinate system origin at the center of the projector exit pupil axes aligned to the projector orientation

t [u v] texture coordinate system used on the unit-distance image plane for each channel (u v) = 0 0 at lower left, 1 1 at upper right

The principal variables vectors and arrays are shown belov Arrays vectors and vector elements of vector arrays ai shown in **bold** and are primed when composed as homogene ous coordinates Subscripts indicate the coordinate systen Scalars are shown in *italic* Orientation vectors (**xxxHPR**) ai composed as (*heading pitch roll*)

dep$_d$ position of the design eyepoint

obs$_d$, chan$_o$, eye$_d$ position of the observer the channel offs from the observer and the final eyepoint for channel

obsHPR$_d$ observer line of sight

chan$_o$ offset for a channel from the eyepoint (almo assuredly zero except perhaps for stereo sep ration of pupils)

chanHPR$_o$ channel orientation from the observer

lfov rfov left right down and up angles of the channel
dfov ufov field of view because of the sign convention for angles for a symmetric field of view *rfov* and *dfov* are negative

left right the x coordinates of the left and right edges
bot top and the y coordinates of the bottom and top edges of the channel s unit distance image plane *left* and *bot* are negative for a symmetric field of view

proj$_d$ position of the projection lens exit pupil

projHPR$_d$ projector orientation

The following are vector elements of arrays for the grid for a given channel ray traced to various surfaces

vert$_{adc}$ points in a user defined grid located on an imaginary unit radius sphere about the design eyepoint

grid$_{ctc}$ points in the grid on a channel s imaginary unit distance image plane

imag$_{cp}$ points projected from the eyepoint through the grid vertices onto the dome surface

prim$_p$ points in a projector s imaginary unit distance image plane

obj$_p$ points in a projector s object plane

led$_{ae}$ tgt$_{ae}$ positions of I EDs and targets on an imaginary unit radius sphere about the design eyepoint

The following are parameters for the dome

N$_d$ Vector from the design eyepoint normal to the flat screen not applicable to curved screens

d g parameters that distinguish between shapes of quadric dome surfaces

R radius of a spherical or cylindrical dome or the minor radius of an ellipsoidal dome

κ plane angle at the apex of a conical dome

The following variables and arrays are used for the lens mapping

focus distance of the focus from the center of an ellipsoidal dome

f focal length of a projection lens

objWidth objHeight dimensions of the object plane of a projector

prImgX **prImgZ** arrays containing x and z coordinates of points on the projector s unit distance image plane used to locate a box of four adjacent entries in the lens mapping table and to set weights for each for the bilinear interpolation

prObjX **prObjZ** mapping tables for the projection lens arrays containing x and z positions on the projector object plane corresponding to the points on the image plane defined by the corresponding entries in **prImgX** and **prImgZ**

> A Note on Linear Units

Linear dimensions and positional data fall into two groups those that apply to positions in the dome and those that apply the projector object plane Items within each group must be in the same units (feet inches millimeters) but the two groups may be in different units

Variables within each group must be in the same linear units	
Group 1	**dep, obs, chan N,** *R focus*
Group 2	*f objWidth objHeight*

> Matrix Functions

Two functions are defined to create transformation matrices for post multiplication of row vectors composed as homogeneous coordinates The functions are

1 **Rotate(HPR)** returns a 4 x 4 rotation matrix for the (*h p r*) contained in **HPR Rotate** is used for point and coordinate transformations as shown in the table below

operation	*point transform*	*coordinate transform*
P Rotate(*h p r*)	*r p h*	*r p h*
P Rotate(*h p r*)	*r p h*	*r p h*
P Rotate^T(*h p r*)	*h p r*	*h p r*
P Rotate^T(*h -p r*)	*h p r*	*-h p r*

2 **Translate(Ofst)** returns a 4 x 4 point translation matrix for the (*x y z*) displacements contained in **Ofst** **Translate(-Ofst)** is the corresponding coordinate trans formation

> Ray Trace Functions

We define a ray trace function to return a scalar value that defines a point on a line where the line intersects the projection surface The function *trace* uses parameters that define the surface in global memory The function is *trace*(**orig dir**) where

orig is a point in dome coordinates (e g the channel eye point **eye$_d$**)

dir is a displacement vector in the direction of interest from **orig** toward the projection surface (e g a point in the channel grid **grid$_c$**)

Then the point of intersection in dome coordinates is

$$\mathbf{IP_d} = \mathbf{orig} + \mathbf{dir}\ trace(\mathbf{orig}, \mathbf{dir})$$

>> Ray Trace for Flat Screens

The parameter for a flat screen is the screen normal vector **Nd**

The parametric form of a line through the point **orig** with direction **dir** is

$$\mathbf{P} = \mathbf{orig} + \mathbf{dir}\, t$$

Note that $t = 0 \Rightarrow \mathbf{P} = \mathbf{orig}$ and $t = 1 \Rightarrow \mathbf{P} = \mathbf{orig} + \mathbf{dir}$

A plane with normal N has the equation

$$(\mathbf{P} - \mathbf{N}) \bullet \mathbf{N} = 0$$

The intersection of the line with the plane is obtained by substitution

$$(\mathbf{orig} + \mathbf{dir}\, t - \mathbf{N}) \bullet \mathbf{N} = 0$$

$$(\mathbf{orig} - \mathbf{N}) \bullet \mathbf{N} + \mathbf{dir}\, t \bullet \mathbf{N} = 0$$

$$trace(\mathbf{orig}, \mathbf{dir}) = \frac{\mathbf{N} \bullet (\mathbf{N} - \mathbf{orig})}{\mathbf{N} \bullet \mathbf{dir}}$$

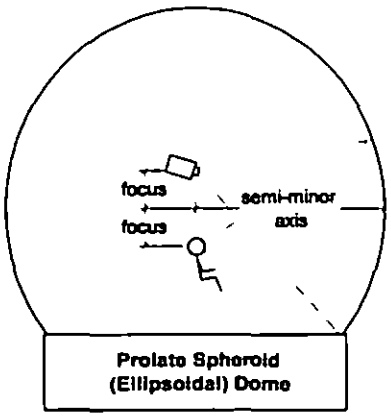
(It should be noted that the performance penalty exacted by distortion correction probably warrants alternative methods for flat screens)

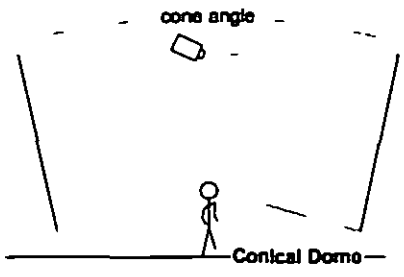
>> Ray Trace for Quadric Screens

Spheres cylinders ellipsoids and cones are quadric surfaces that can all be represented by the form

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{kz^2}{c^2} - d = 0$$

Ellipsoidal and conical screens shown notionally at right, offer improved bend angles compared with spheres and cylinders resulting in higher brightness and contrast when used with gain coatings Cylindrical and conical domes are used in particular in ship s bridge simulators where the vertical field of view above the horizon is typically small





Subject to the practical assumptions for domes that *i)* the surfaces are rotationally symmetric about z and *ii)* an ellipsoid is a prolate spheroid (an ellipse rotated about its major axis) then for these common dome shapes $a^2 = b^2$ and the defining equation can be reduced to $x^2 + y^2 + gz^2 - d = 0$

Parameter values for g and d are shown below

Shape and Orientation	g	d
Sphere center at origin	1	R^2
Ellipsoid center at origin foci at $[0 \;\; 0 \;\; \pm focus]$	$\frac{1}{1 + (focus / R)^2}$	R^2
Cone vertex at origin axis on z	$-\tan^2 \kappa/2$	0
Cylinder axis on z	0	R^2

The ray trace onto the dome is performed by substituting th parametric expression for the line into the dome equation

$$(orig_x + dir_x\, t)^2 + (orig_y + dir_y\, t)^2 + g(orig_z + dir\; t)^2 - d = 0$$

$$\underbrace{(dir_x^2 + dir_y^2 + g\, dir^2)}_{p} t^2 + \underbrace{2(orig_x dir_x + orig_y dir_y + g\, orig\, dir_z)}_{q} t + \underbrace{(orig_x^2 + orig_y^2 + g\, orig_z^2) - d}_{r \text{ (compute once per channel)}} = 0$$

Then $trace(\mathbf{orig}, \mathbf{dir}) = \frac{-q + \sqrt{q^2 - 4pr}}{2p}$

(The positive sign for the quadratic determinant chooses th intersection from **orig** toward **dir**)

>> Ray Tracing for Other Surfaces

The surfaces described are those familiar to the author Ra trace functions for other surfaces (e g the occasional torus could readily be developed by the ambitious implementer

Real-Time Processing

Processing is generally into those tasks that must be pe formed in the Application process and those that are pe formed in the Draw process

> Application

The ray trace is performed in the Application process Fo static systems this need only be performed once for dynami systems it must be performed every frame or as often as any thing relevant changes

1 Form a coordinate transformation matrix to translate the channel offset from observer to dome coordinates

$$\mathbf{Transform}_{od} = \underbrace{\mathbf{Rotate}^{T}(\mathbf{obsHPR}_{d})}_{\text{rotate orientation from observer to dome}}\ \underbrace{\mathbf{Translate}(-\mathbf{obs}_{d})}_{\text{translate origin from observer to dome}}$$

2 For *each projector* for *each channel*

a Locate the eyepoint in dome coordinates

$$\mathbf{eye}_{d} = \mathbf{obs}_{d} + \underbrace{\mathbf{chan}_{o}}_{\text{channel offset}}\ \underbrace{\mathbf{Transform}_{od}}_{\text{transform coordinates from observer to dome}}$$

b Form a coordinate transformation matrix from eye to projector coordinates

$$\mathbf{Transform}_{ep} = \underbrace{\mathbf{Translate}(\mathbf{eye}_{d} - \mathbf{proj}_{d})}_{\text{translate origin from eyepoint to projector}}$$

$$\underbrace{\mathbf{Rotate}^{T}(\mathbf{projHPR}_{d})}_{\text{rotate cordinates from eyepoint to projector}}$$

c Form a point transformation matrix to position the channel grid in eye coordinates

$$\mathbf{Transform}_{ce} = \underbrace{\mathbf{Rotate}(\mathbf{chanHPR}_{o})}_{\text{rotate point from channel to observer}}$$

$$\underbrace{\mathbf{Rotate}(\mathbf{obsHPR}_{d})}_{\text{rotate point from observer to dome}}$$

d If the grid for the channel is static (described later) then $\mathbf{grid}_{c}$ computed initially is still valid If the channel grid is dynamic get the field of view angles for the current frame and recompute the grid locations

The bounds of the unit distance image plane for the each channel are

$$top = \tan ufov$$

$$left = -\tan lfov \quad right = -\tan rfov$$

$$bot = \tan dfov$$

And the grid in channel coordinates is

for $row = 0\ rows-1 \quad col = 0\ cols-1$

$$\mathbf{grid}_{c} = \left[col\frac{right-left}{cols-1}+left \quad row\frac{top-bot}{rows-1}+bot\right]$$

e For *each point* in the grid

i Transform the grid from channel coordinates to eye coordinates

$$\mathbf{grid}'_{e} = \mathbf{grid}_{c}\ \mathbf{Transform}_{ce}$$

ii Ray trace the points on the grid to the dome surface in eye coordinates

$$\mathbf{imag}_{e} = \mathbf{grid}_{e}\ trace(\mathbf{eye}_{d}, \mathbf{grid}_{e})$$

iii Transform the points in the image from eyepoint to projector coordinates

$$\mathbf{imag}_{p} = \mathbf{imag}_{e}\ \mathbf{Transform}_{ep}$$

iv Transform the points in the image to the projector s unit distance image plane by a perspective division

$$\mathbf{prim}_{p} = \frac{\mathbf{imag}_{p}}{imag_{pn}}$$

v Map the point in the projector s unit-distance image plane to the projector object plane by bilinear interpolation[4] of the lens mapping table for the projector (the derivation of the mapping table is described later)

$$obj_{px} = \text{BilInterp}\left[\mathbf{prim}_{p}\ \mathbf{prImgX}\ columns \atop \mathbf{prImgZ}, rows\ \mathbf{prObjX}\right]$$

$$obj_{pz} = \text{BilInterp}\left[\mathbf{prim}_{p}\ \mathbf{prImgX}\ columns \atop \mathbf{prImgZ}, rows\ \mathbf{prObjZ}\right]$$

> Draw

In the Draw process all channels are rendered in normal perspective fashion At the conclusion of the usual Draw each channel is represented as an image in the frame buffer The perspective image is copied from frame buffer to texture memory and applied as a texture to the triangular mesh whose vertices were computed in the App process The mesh is rendered in the orthographic output channel for the projector

1 For *each projector* for *each channel*

a Compute the grid locations using one of the methods described later (the channel grid may change if the field of view is varied)

b Get the channel pixel dimensions *chanWid* and *chanHt* (The size may vary from the initial size if the window or channel is dynamically resized)

c Copy the active image from frame buffer to the allocated texture in texture memory

d Recalculate the texture indices of the grid based on the channel dimensions and the size of the channel s allocated texture (*texWid* and *texHt*)

$$\mathbf{grid}_{t} = \begin{bmatrix} \frac{texWid}{chanWid}\frac{grid_{xc}-left}{right-left} \\ \frac{texHt}{chanHt}\frac{grid_{c}-bot}{top-bot} \end{bmatrix}^{T}$$

e Render the grid in the projector s orthographic output channel The vertices are in $\mathbf{obj_p}$ and the corresponding texture coordinates are in $\mathbf{grid_t}$

Grid Definitions

Each channel must be divided into a grid It is the points in this grid that are ray traced through several stages to compute

---

[4] See *Numerical Recipes in C* William H Press Cambridge University Press 1992 for a bilinear interpolation routine

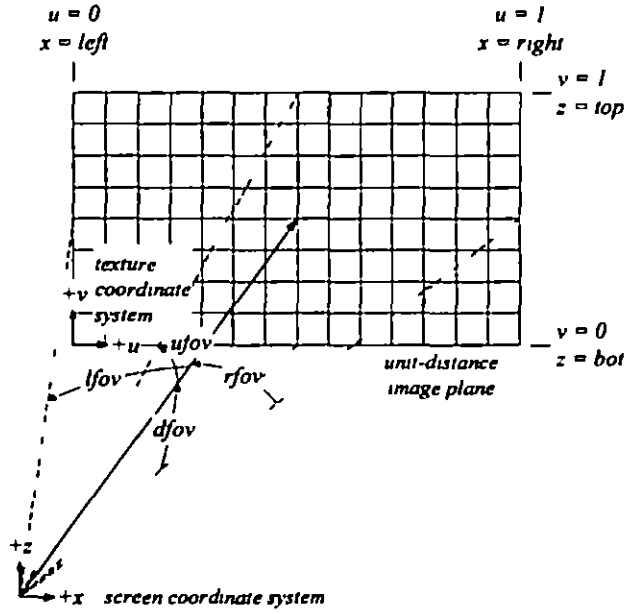
the final mesh on which the perspective textures are applied to form the pre-distorted image There are various methods to define a grid with an appropriate choice based on channel configuration mosaicing and edge blending requirements

Each channel s grid may be either static (computed once dur ing initialization) or dynamic (recomputed as required to ac commodate changes in the channel field of view) Dynamically resizing the channel does not effect the grid and therefore does not influence the decision to make the grid dynamic or static

> Regular Grid

A regular grid is useful when a projector s rectangular chan nels tessellate the allocated field of view It divides the channel into a regular rectilinear grid of specified density in the u and v directions Each point in the grid is defined by texture coordinates ($grid_{tu}$ $grid_{tv}$) with values from (0 0) at the lower left to (1 1) at the upper right of the channel

The figure below shows a 15 x 9 grid for the SOPS back ground channel that results in the distorted mesh shown in the previous figure



The bounds of a channel s unit-distance image plane are

$$top = \tan ufov$$

$$left = -\tan lfov \quad right = -\tan rfov$$

$$bot = \tan dfov$$

Then the grid in channel coordinates is

$$\text{for } row = 0 \; rows - 1 \quad col = 0 \; cols - 1$$

$$\mathbf{grid}_c = \left[col \frac{right - left}{cols - 1} + left \quad 1 \quad row \frac{top - bot}{rows - 1} + bot \right]$$

> Spherical Grids

Spherical grids are convenient when channels must be cut along borders other than the channel edges to tile the total field of view Three types are summarized below

Zone This is a standard lat/long graticule and is the grid used in a Hammer Aitoff plot The angles are *pitch* p $+y$ into $+z$ followed by *heading* h $+y$ into $-x$ and the transformation to rectangular coordinates is

$$\mathbf{vert}_d = [-\sin\lambda\cos\varphi \quad \cos\lambda\cos\varphi \quad \sin\varphi]$$

Mosaic This is just a zone grid rotated forward It is useful when multiple channels mosaiced into a hemisphere like Hemi HIROPS or for target projectors with circular aperture masks The angles are *colatitude* ρ $+y$ into $+x$ followed by *roll* r $+x$ into $-z$ The transformation is

$$\mathbf{vert}_d = [\sin\rho\cos r \quad \cos\rho \quad -\sin\rho\sin r]$$

Lune Great-circle graticule useful for rectangu lar channels like SOPS or for target projectors The angles are *heading* h $+y$ into $-x$ followed by *pitch* p $+y$ into $+z$ and the transformation is

$$\mathbf{vert}_d = [-\sin h \quad \cos h\cos p \quad \cos h\sin p]$$

The transformation of spherical grid vertices to grid points i channel coordinates for *each channel* is as follows

1 Compute a coordinate transformation matrix from dom to channel coordinates for the vertices

$$\mathbf{Transform}_{dc} = \underbrace{\underbrace{\mathbf{Translate}[-\mathbf{obs}_d]}_{\text{translate origin from dome to observer}} \underbrace{\mathbf{Rotate}^T[\mathbf{obsHPR}_d]}_{\text{rotate coordinates from dome to observer}}}_{\text{transform from dome to observer coordinates}}$$

$$\underbrace{\underbrace{\mathbf{Translate}[-\mathbf{chan}_c]}_{\text{translate origin from observer to channel}} \underbrace{\mathbf{Rotate}^T[\mathbf{chanHPR}_c]}_{\text{rotate coordinates from observer to channel}}}_{\text{transform from observer to channel coordinates}}$$

2 For *each vertex*

a Transform the vertex to homogeneous Cartesian dom coordinates using one of the transformations shown t compute $\mathbf{vert}_d$

b Transform the vertex from dome to channel coord nates

$$\mathbf{vert}'_c = \mathbf{vert}'_d \, \mathbf{Transform}_{dc}$$

c Project the vertex onto the channel s image plane b perspective division

$$\mathbf{grid}_c = \frac{\mathbf{vert}_c}{vert_{cy}}$$

User vertices that follow great circles of the unit sphere (e g lines of longitude) will always trace straight lines on th channel, those that follow minor circles (e g lines of lat tude other than the equator) will generally trace curved lin on the channels

Lens mapping

Points are transformed from the projector s unit distance image plane to the projector s object plane by bilinear inter polation of two mapping tables **prObjX** and **prObjZ** Tables are used because they accommodate arbitrary lens mappings including off axis optics and because they readily allow in corporation of error terms derived from the alignment procedure

The transformation from the projector s image plane to the object plane is

$$obj_{px} = \text{BilInterp}\left[\begin{matrix}\mathbf{prim}_p & \mathbf{prImgX}\ columns \\ \mathbf{prImgZ}, rows & \mathbf{prObjX}\end{matrix}\right]$$

$$obj_{pz} = \text{BilInterp}\left[\begin{matrix}\mathbf{prim}_p & \mathbf{prImgX}\ columns \\ \mathbf{prImgZ}, rows & \mathbf{prObjZ}\end{matrix}\right]$$

The mapping tables may be derived by any of the several means described below

> Header row and column for mapping tables

The row and column vectors **prImgX** and **prImgZ** support the interpolation of the mapping tables **prObjX** and **prObjZ** The mapping tables each contain as many columns as **prImgX** and as many rows as **prImgZ** The contents of these arrays **prImgX** and **prImgZ** are x and z positions on the unit distance image plane of the projector For a given loca tion on the projector image plane each array is searched to find a pair of entries that bracket the x or z position The rela tive proximity of the search value to the upper and lower bounds in the array are used to derive weights for the bilinear interpolation

These two arrays are created in the same fashion for the f tan theta f theta or User l ist mapping functions The computa tion is shown below The mapping tables must be sufficiently dense to keep interpolation errors within acceptable limits

$$maxAngle = 89.9$$

$$prImg\lambda_{j=1\ mapSize} = \tan\left[\frac{2\left[j-1\right]maxAngle}{mapSize-1} - maxAngle\right]$$

$$prImgZ_{i=j} = prImg\lambda_j$$

> f-tan theta

An f tan theta lens is the nominally distortionless lens com monly used in cameras and CRT projectors The f tan theta mapping tables are derived as shown below As the equation shows the f tan theta mapping is a simple linear scaling of the projector image plane to the object plane by the focal length f

for $i = 1\ mapSize\ \ j = 1\ mapSize$

$$prObj\lambda_{i,j} = f\ prImg\lambda_j$$

$$prObjZ_{i,j} = f\ prImgZ_i$$

> f-theta

In an f theta lens magnification decreases with increasing field angle so that an entire hemisphere in image space can be mapped onto the object plane (In principal an f theta lens could map the entire sphere about the exit pupil onto the object plane but the plane mapping technique restricts the field of the lens to a hemisphere Note that this limitation applies to field angles about the exit pupil not the eyepoint and does not affect compatibility with any known deployed display system)

for $i = 1\ mapSize\ \ j = 1\ mapSize$

$$imageHeight = \sqrt{prImg\lambda_j^2 + prImgZ_i^2}$$

if $imageHeight = 0$ then $prObj\lambda_{i,j} = prObjZ_{i,j} = 0$ else

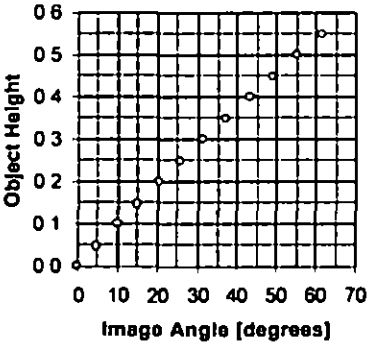
$$imageAngle = \arctan\left[imageHeight\right]$$

$$prObj\lambda_{i,j} = f\ imageAngle\frac{prImg\lambda_j}{imageHeight}$$

$$prObjZ_{i,j} = f\ imageAngle\frac{prImgZ_i}{imageHeight}$$

> User List

A user list is a method of deriving mapping tables for a lens that is axially symmetric but not adequately described as either f theta or f tan theta The user list is a tabulation of exit pupil angle versus object plane height For exam ple the plot at right shows data for a hypo thetical f theta lens with yet more barrel distor tion The mapping tables are computed as shown below



LinInterp is a linear interpolation function

for $i = 1\ mapSize\ j = 1\ mapSize$

$$imageHeight = \sqrt{prImg\lambda_j^2 + prImgZ_i^2}$$

if $imageHeight = 0$ then $prObjX_{i,j} = prObjZ_{i,j} = 0$

else

$$imageAngle = \arctan\left[imageHeight\right]$$

$$objectHeight = \text{LinInterp}\left[list\ length\ imageAngle\right]$$

$$prObj\lambda_{i,j} = prImg\lambda_j\frac{objectHeight}{imageHeight}$$

$$prObjZ_{i,j} = prImgZ_i\frac{objectHeight}{imageHeight}$$

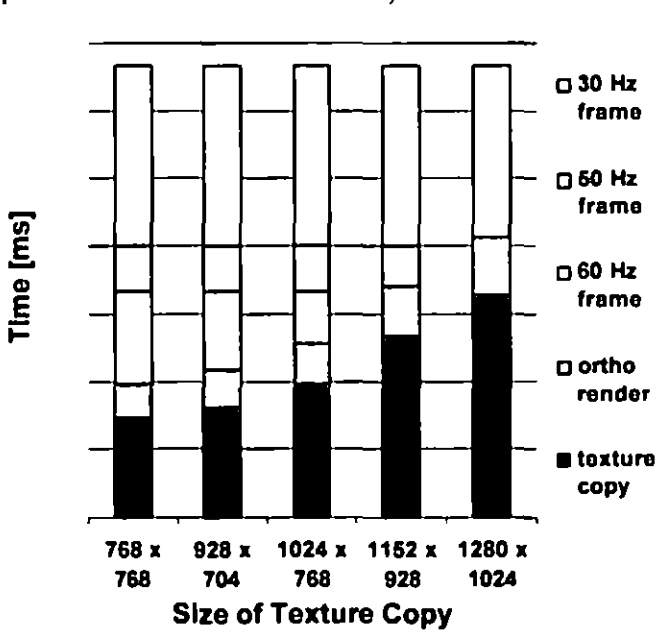
> User Map

For tilted optics (lenses that are not rotationally symmetric as is the case with Hemi HIROPS) users may create their own

mapping tables from optical ray tracing of the projection lens and use them directly

Dynamic Video Resizing

Performance of applications employing distortion correction will depend significantly on the size selected for the perspective channels because the copy time from frame buffer to texture memory alone consumes a significant portion of the frame The ability to dynamically resize the perspective channels provides a strongly levered means of load management The plot below shows a bar graph of approximate timing on an SGI Onyx2 IR how a frame (60 50 or 30 Hz) might be spent for the texture copy and rendering of the orthographic output channel The time remaining in the frame is the time available for drawing the perspective channels (SGI has announced plans for a multiplexer that enables two or more pipes to be cascaded to a single video output, enabling 60 Hz update rate at 1280x1024 resolution)



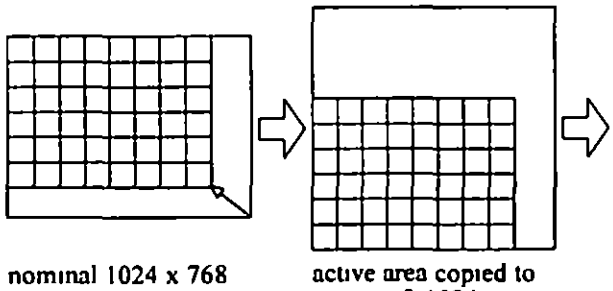
The plot shows two basic perspective channel sizes (1024 x 768 and 1280 x 1024) and two channels that are slightly reduced from these sizes Note the significant difference in frame time available for the perspective draw as a result of only small reductions in channel size

This is similar to the Onyx2 s DVR feature, by which the *infiniteReality* graphics system can dynamically reduce the resolution of rendered imagery, while maintaining a constant video output format Reducing resolution reduces pixel fill demand to maintain frame rate in scenes with high depth complexity Maintaining a constant video output format is required so as not to disturb the synchronization of the projector and is performed in the *infiniteReality* Display Generator hardware by bilinear interpolation of the reduced image

In the first (perspective) rendering pass, a channel is rendered at some nominal resolution say 1024 x 768 Then the image in the frame buffer is copied to texture memory Since textures in Open GL must be powers of two in both dimensions the image is copied into a larger 1024 x 1024 texture map The top and right portions of the texture map are unused (i e no (u v) coordinates are mapped into it) If the perspective channel has been reduced in size only the rendered image is copied to texture memory and the (u v coordinates are scaled so as to reference only the active portion of the texture for the current frame Copying only the active image reduces the time required

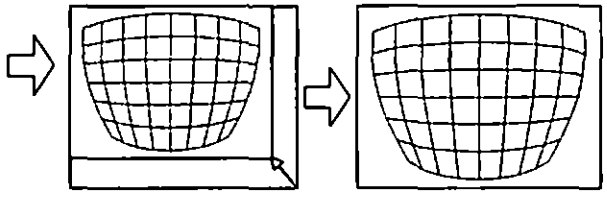
In the second (orthographic) rendering pass the distortion mesh is rendered at some nominal resolution and the texture computed in the first pass is applied DVR may also be used in this pass Note that there is only a single texture map used so texturing is performed perforce using bilinear interpolation rather than the trilinear method used when MIP maps are available In general the texture will be minified in some areas of the output channel and magnified in others The user should consider the relationship between the texture dimensions and the dimensions of the output channel both in setting initial parameters and in exercising dynamic control of channel size to minimize artifacts due to excessive magnification or minification

This process is summarized in the figure below



nominal 1024 x 768 perspective image minified by application

active area copied to corner of 1024 x 1024 texture (u v)'s in original grid remapped to active area of texture



texture applied to mesh via bilinear interpolation in RM to nominal 1000 x 800 ortho channel, also may be dynamically minified

active area displayed via bilinear interpolation in the DG as final 1024 x 768 image

DVR *per se* is a hardware feature supported currently only on the SGI IR, but the more strongly levered method of reducing perspective channel size is possible on any platform

Alignment

> Rationale

Despite the fundamentally rigorous approach described here the image will still have residual geometric distortion for one or more of the reasons described below The alignment proc ess described here can reduce distortion to well within accept able values provided the basic assumptions are correct (the dome surface is nominally described by the surface selected the mapping tables are basically correct and adequately dense the projector exit pupil location is reasonably correct) The alignment process aligns the image to the dome surface by modifying the lens mapping tables although the mapping tables may already be a perfect description of the lens This approach resolves several possible sources of error some more likely than others as discussed below

What are the possible sources of the residual distortion and how is it best corrected?

- **LEDs** The azimuths and elevations of the LED fiducials about the eyepoint might not be correct This is discounted as a likely source of error however since the LED posi tions are located with a theodolite (an extremely accurate instrument) in the as built dome The LEDs are regarded as perfectly correct in *az* and *el*

Dome Despite being at the correct (*az el*) the LEDs may not be at the correct locations in space as a result of errors in the dome surface Aside from incorrectly specifying the general surface (e g sphere versus ellipsoid) the dome surface offers two other potential sources of error

  - The dome surface may be somewhat larger or smaller than its design size To avoid this causing a problem the dome should be measured in the as built configura-tion and the best fit actual value(s) used in specifying dome parameters
  - All domes are somewhat irregular in shape geometry errors will be most pronounced where the surface tan gent is most oblique to the projector or design eye point For a stationary projector modifying the lens map is a clean solution For a servo-controlled projec tor the only solution is to correct the lens map for the design basis case and accept the errors that occur with different projector angles In principal this could be fixed by treating the dome as a polyhedral surface for the ray trace

- **Raster** There are several sources of error in the source image for the projection optic including horizontal and vertical size and position raster rotation and whatever geometric distortion exists in the raster All of these sources of error are accurately corrected by changing the lens mapping provided that the mapping of the image onto the object plane doesn t change with rotations of the lens of a servo-controlled projector

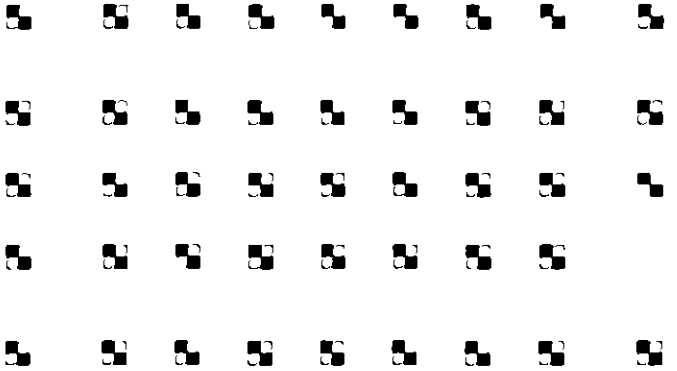
- **Optics**
  - The position of the projector exit pupil may be incor rectly specified
  - The specified mapping function may be inaccurate i e the lens may depart from its nominal f theta or f tan theta function or the user list or user map func tion may be imperfect These of course are directly addressed by changing the lens mapping function
  - For a servo-controlled projector if rotating compo nents of the optical train are off center (misalignment in the optical train) the image will nutate with rotation The image may be statically corrected in the fixed forward position by changing the lens mapping but geometry errors will vary with projector line of sight

The conclusion is that the likely set of correctable errors is best addressed by correcting the mapping function

> Method

The alignment process described here allows users to perform fine alignment of the image to the dome surface Alignment for servo controlled projectors is performed with the projector in the design-basis position Alignment is performed sepa rately for each projector

LED positions specified in *az* and *el* about the design eye point are read from a file A test pattern is rendered consist ing of a square target positioned on the surface of a sphere at the azimuth and elevation of each LED One target at a time is highlighted as shown in the figure below



Each target is moved in azimuth and elevation to center it over the corresponding LED The position of the LEDs and the displaced targets are ray traced to the dome then to the projector image plane and then mapped to the projector object plane The object plane position of the LEDs and the object plane offsets of the targets are saved in an array

OfstList (*LEDName* $ledObj_{px}$ $ledObj_{pz}$ $ofst_{px}$ $ofst_{pz}$)

where $ledObj_{px}$ $ledObj_{pz}$ is the nominal position of the LED on the projector object plane

$ofst_{px}$ $ofst_{pz}$ is the displacement on the projector object plane required for the target to superim pose the corresponding LED

N is the total number of LEDs

Then a polynomial fit is computed for the offsets as a function of the object plane position of the LED The polynomials (one for x and one for z) are used to compute a correction for the lens maps

> Ray tracing the LEDs and targets

1 Form a transformation matrix from dome to projector coordinates

$$\mathbf{Transform}_{dp} = \mathbf{Translate}(-\mathbf{proj}_d)\,\mathbf{Rotate}^T(\mathbf{projHPR}_d)$$

2 For *each LED*

a Locate the position of each LED on the imaginary unit radius sphere about the design eyepoint (note that *az* is measured as a compass angle clockwise from north – the opposite sign convention as from *heading*)

$$\mathbf{led}_e = \left[\sin led_{az}\cos led_{el} \quad \cos led_{az}\cos led_{el} \quad \sin led_{el}\right]$$

b Ray trace the LED from the design eyepoint onto the surface of the dome

$$\mathbf{ledImag}_d = \mathbf{dep}_d + \mathbf{led}_e\ trace(\mathbf{dep}_d, \mathbf{led}_e)$$

c Transform to projector coordinates

$$\mathbf{ledImag}_p = \mathbf{ledImag}_d\ \mathbf{Transform}_{dp}$$

d Perspective divide to project onto the unit-distance plane

$$\mathbf{ledPrim}_p = \frac{\mathbf{ledImag}_p}{ledImag_{py}}$$

e Map to the projector object plane and normalize to a unit width object plane

$$ledObj_{px} = \frac{\mathrm{BilInterp}\left(\begin{matrix}\mathbf{ledPrim}_p\ \mathbf{prImgX}\ columns \\ \mathbf{prImgZ}, rows\ \mathbf{prObjX}\end{matrix}\right)}{objWidth}$$

$$ledObj_{pz} = \frac{\mathrm{BilInterp}\left(\begin{matrix}\mathbf{ledPrim}_p\ \mathbf{prImgX}\ columns \\ \mathbf{prImgZ}, rows\ \mathbf{prObjZ}\end{matrix}\right)}{objHeight}$$

3 For *each target* in identical fashion

$$\mathbf{tgt}_e = \left[\sin tgt_{az}\cos tgt_{el} \quad \cos tgt_{az}\cos tgt_{el} \quad \sin tgt_{el}\right]$$

$$\mathbf{tgtImag}_d = \mathbf{dep}_d + \mathbf{tgt}_e\ trace(\mathbf{dep}_d, \mathbf{tgt}_e)$$

$$\mathbf{tgtImag}'_p = \mathbf{tgtImag}_d\ \mathbf{Transform}_{dp}$$

$$\mathbf{tgtPrim}_p = \frac{\mathbf{tgtImag}_p}{tgtImag_{py}}$$

$$tgtObj_{px} = \frac{\mathrm{BilInterp}\left(\begin{matrix}\mathbf{tgtPrim}_p\ \mathbf{prImgX}, columns \\ \mathbf{prImgZ}, rows\ \mathbf{prObjX}\end{matrix}\right)}{objWidth}$$

$$tgtObj_{pz} = \frac{\mathrm{BilInterp}\left(\begin{matrix}\mathbf{tgtPrim}_p\ \mathbf{prImgX}\ columns \\ \mathbf{prImgZ}\ rows\ \mathbf{prObjZ}\end{matrix}\right)}{objHeight}$$

4 For *each LED* and its corresponding target, find the objec plane offset

$$\mathbf{ofst}_p = \mathbf{tgtObj}_p - \mathbf{ledObj}_p$$

> Polynomial Correction

After centering each target over its corresponding LED th task remains to build new mapping tables **prObjX** an **prObjZ** from the original maps and **OfstList** This is per formed by computing coefficients for the polynomials *xOfs* and *zOfst* which are least squares fits to the x and z displace ments The polynomial functions enable an extrapolation o the discrete list of target offsets to offsets for every location i tables **prObjX** and **prObjZ**

The method described here performs up to a fifth-order fi the user must select an appropriate subset of the terms depending on the number of LEDs available The number o LEDs should be three or more times greater than the numbe of polynomial terms used

The general fifth-order two-dimensional function for *xOfst* is

$$xOfst = \begin{bmatrix}1\\x\\x^5\end{bmatrix}^T \begin{bmatrix}b_{x00} & b_{x01} & & b_{x05}\\ b_{x10} & & & \\ & & & \\ b_{x50} & & & b_{x55}\end{bmatrix} \begin{bmatrix}1\\z\\z^5\end{bmatrix} = \sum_{i=0}^{5} x^i \left[\sum_{j=0}^{5} b_{ij}\, z^j\right]$$

The arguments to the polynomial are normalized positions o the pre-corrected projector object plane the value of the poly nomial is the x offset on the normalized object plane The co responding function for *zOfst* is exactly symmetric in x and z

The coefficients b_{x00} through b_{x35} for *xOfst* are computed result in a least squares polynomial function fit to the *N ofs* entries in **OfstList** by solving a system of 36 norm equations for each polynomial **OX BX** = **GX** for *xOfst* an **OZ BZ** = **GZ** for *zOfst* **BX** and **BZ** are the polynomi coefficients to be solved in the form of 36 x 1 colum matrices

$$\mathbf{BX}(36x1) = \begin{bmatrix}b_{x00}\\b_{x01}\\ \\b_{x55}\end{bmatrix} \qquad \mathbf{BZ}(36x1) = \begin{bmatrix}b_{x00}\\b_{x01}\\ \\b_{x55}\end{bmatrix}$$

OX and **OZ** are square matrices formed from the object plane positions of the LEDs **OX** is shown below **OZ** is exactly symmetric in x and z

$$ox_{i=0\ 35\ j=0\ 35} = \sum_{n=1}^{k} ledObj_{xn}^{int(i/6)\ int(j/6)} ledObj_{n}^{mod(i\ 6)\ mod(j\ 6)}$$

GX and **GZ** are column matrices formed from the object plane positions of the LEDs and the object plane offsets required to center the targets over the LEDs **GZ** is again symmetric in x and z

$$gx_{i=0\ 35} = \sum_{n=1}^{k} ledObj_{xn}^{int(i/6)}\ ledObj_{.n}^{mod(i\ 6)}\ ofst_{xn}$$

Then the solution for the coefficients is just $\mathbf{BX} = \mathbf{OX}^{-1}\,\mathbf{GX}$

The polynomials are then used to compute offsets for each location in the mapping tables

for $i = 1\ mapSize\ j = 1\ mapSize$

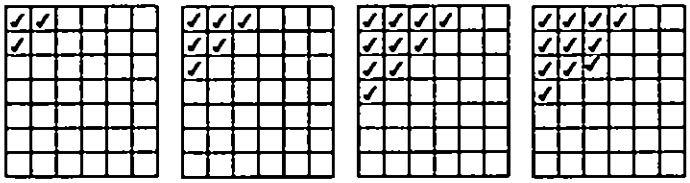
$$prObjX_{i\,j} = prObjX_{i\,j} + objWidth\ xOfst\left\{\frac{prObjX_{i\,j}}{objWidth}\ \frac{prObjZ_{i\,j}}{objHeight}\right\}$$

$$prObjZ_{i\,j} = prObjZ_{i\,j} + objHeight\ zOfst\left\{\frac{prObjX_{i\,j}}{objWidth}\ \frac{prObjZ_{i\,j}}{objHeight}\right\}$$

The offsets are thus added to corresponding entries in the original mapping tables to create the corrected tables

>> Solution using decimated terms

The use of the most possible number of terms (i e one term for every LED) of the general fifth-order polynomial will always result in an equal or better fit of the targets to the LEDs than any solution using decimated terms However it will also result in disastrous local geometry errors between LEDs This is because the interpolating polynomial will have large and nearly canceling coefficients for high degree terms that achieve precise matching at the tabulated points while oscillating wildly in between Some suggested selections of terms are shown in the table Users should consider the types of distortion likely to occur in their projection geometry and then use only those terms in the polynomials As a point of departure users may consider using terms in the following order



position size tilt (9+ LEDs)	+linearity trapezoid curvature (21+ LEDs)	+edge comp edge tilt, pin cushion s tilt (30+ LEDs)	+edge curvature (33+ LEDs)

Observing the fit of the targets to the LEDs as well as the overall shape of the grid will ease the task of selecting appropriate terms

Notes

1 LEDs should be denser in critical areas The LEDs need not be in a regular array or even symmetric

2 There should be about three or more times as many LEDs as the number of polynomial terms used

3 There are two sources of interpolation error that will affect the ability to align a projector

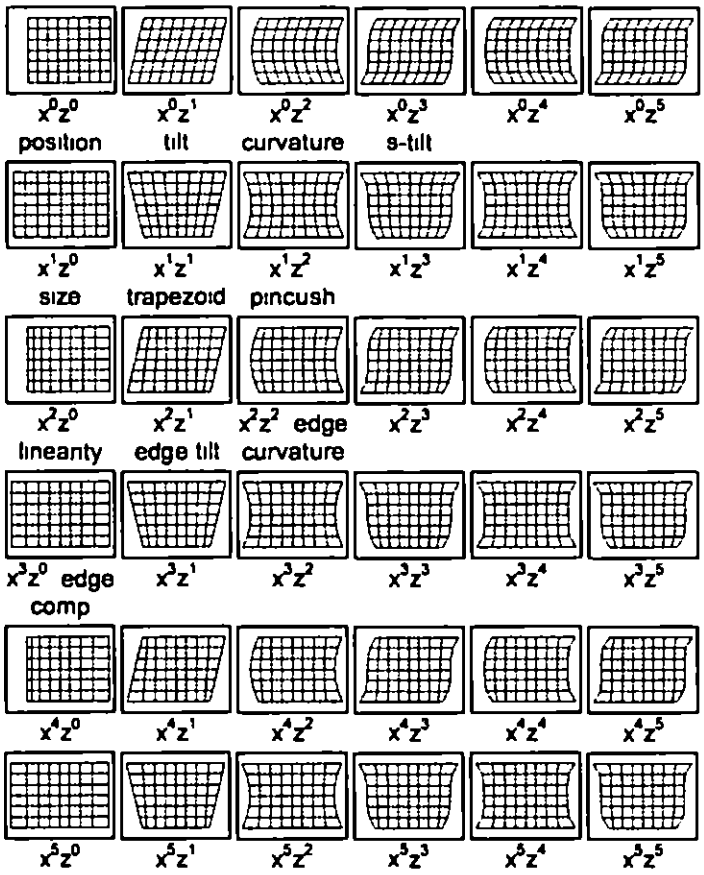
- The lens mapping table is interpolated to map grid vertices to the projector object plane If it is too sparse vertices will not placed with adequate accuracy
- The texture is stretched bilinearly between grid points If the grid is too sparse the interpolation will be an inadequate approximation of the true position

In either case the offsets required to center the LEDs over the target will not be correct and neither therefore will the polynomial coefficients The solution is to make sure that *mapSize* is adequately dense if a highly distorting lens is used and set grid density overly large (e g 50 x 50) during alignment The grid density may be reduced to lower values which still yield acceptably small local geometry errors for real time operation

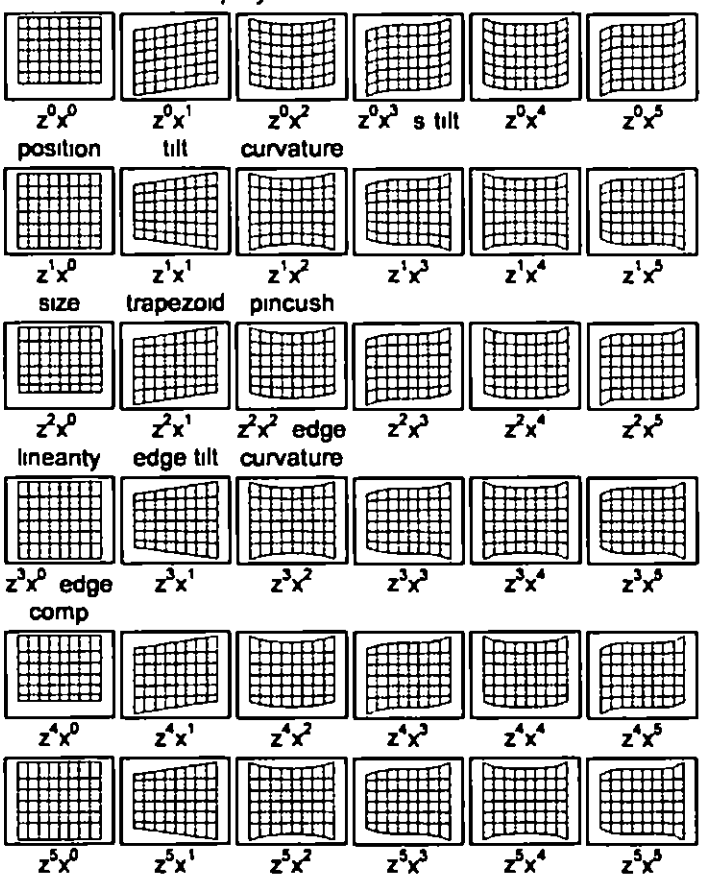
4 Ensure that all of the targets are visible in the projector output before correcting the lens map any targets that are not visible implicitly have zero offsets and will incorrectly affect the alignment

5 The two tables following show the effect of each of the polynomial terms Eleven of the lower power coefficients of the polynomial correspond to the algebraic raster shaping adjustments common to CRT projectors and are shown with their common name The other terms are higher-order variants Each is shown for a coefficient value of +0 15

Effect of polynomial coefficients on x axis



Effect of polynomial coefficients on z axis



Acknowledgements

The original demonstration of dome distortion correction o
Silicon Graphics workstations was developed by ANgu
Henderson then with SGI and now happily with Paradigm
Paradigm s Doug Price coded all of the algorithms here an
developed several innovative tools to assist in configuratior
edge blending and alignment

Real Time Image Visualization for Sensors

David C Anding
Photon Simulations Inc
5720 Oberlin Drive San Diego CA 92121

and

Alexander Szabo
Szabo Software Inc
1325 East 100 South Street Logan UT 84321

Keywords Image Visualization Synthetic Environments Real Time Simulation Infrared Sensors

ABSTRACT

Real time image visualization simulation for sensors operating against synthetic environments comprised of natural backgrounds cultural features mobile objects and dynamic weather is now a reality A commercial software product is available which is capable of providing sensor image visualizations for any spectral filter from the visible through the far infrared The product is called SensorVision™ and is a module of a product called Vega™ It is built upon IRIS Performer™ and OpenGl ™ software and is targeted for use on Silicon Graphics Onyx™ computers with InfiniteReality™ or RealityEngine2™ graphics hardware Vega with SensorVision is ideally suited to provide the scene image input to a real time hardware in the loop sensor simulation ranging from image intensified night vision goggles to midwave FLIRs to longwave Fl IRs SensorVision images are quantitative (each image pixel is expressed in watts/cm^2/steradian) are computed in real time and represent the diurnal effects of weather (including surface temperature variation) on scene images

This paper presents the radiometric processes and algorithms used by the software when computing its output images and discusses the use of the software in hardware in the loop simulation The paper also highlights software capabilities and features e g

- Images include reflection from sun/moon and ambient sky illumination and thermal emission from extended polygons with radiometric shading between vertices and atmospheric attenuation and path radiance with pixel line of sight variability
- Polygon surface temperatures of natural backgrounds and cultural features are computed asynchronously and continuously updated throughout diurnal cycle
- Polygons are radiometrically textured and spatially correlated with visible RGB textures

1 0 INTRODUCTION

Image visualization systems are commonplace in the DoD community and their state of the art is growing rapidly These systems range from visual displays of a synthetic environment (e g the classic RGB displays seen in training simulators) to monochrome views of the same environments as seen by sensors having spectral responses ranging from the visible to the IR Whether a visual view or sensor view amongst their applications are providing image input to training and sensor performance simulators as well as input to photon stimulators for hardware in the loop test and evaluation

SensorVision is a COTS image visualization system for sensors It provides image visualizations for any spectral filter from the visible through the far infrared and for sensors operating against synthetic environments comprised of natural backgrounds cultural features mobile objects weapon effects and dynamic weather Given a time variable position heading pitch and roll of a viewing sensor SensorVision computes and outputs quantitative line of sight radiance (watts/cm^2/sr) for each pixel of at aperture sensor images of a viewed environment The environment may be either static or dynamic in its state description and images can be generated at the rates of sensor perception The COTS system consists of five separate software libraries OpenGL™ IRIS Performer™ Vega™ VR Link™ and SensorVision™ It operates on Silicon Graphics Onyx™ computers with InfiniteReality™ or RealityEngine2™ graphics hardware OpenGl and IRIS Performer are products of Silicon Graphics Inc VR

Link is a product of MaK Technologies, Inc., and Vega is a product of Paradigm Simulation, Inc. Only the details of SensorVision are given in this paper. The reader is referred to SGI documentation for descriptions of their computers and OpenGL and IRIS Performer. References 1and 2 through 4 provide descriptions of VR-Link and Vega, respectively.

2.0 SENSORVISION DESCRIPTION

SensorVision is comprised of a run-time library plus two database construction tools, MOSART Atmospheric Tool (MAT) and Texture Material Mapper (TMM). The production of SensorVision imagery involves steps and processes shown in Figure 1. SensorVision is built on top of Vega, IRIS Performer, and OpenGL. In its present form it only executes on SGI computers and only achieves video framing rate performance on computers with InfiniteReality or RealityEngine2 graphics hardware.

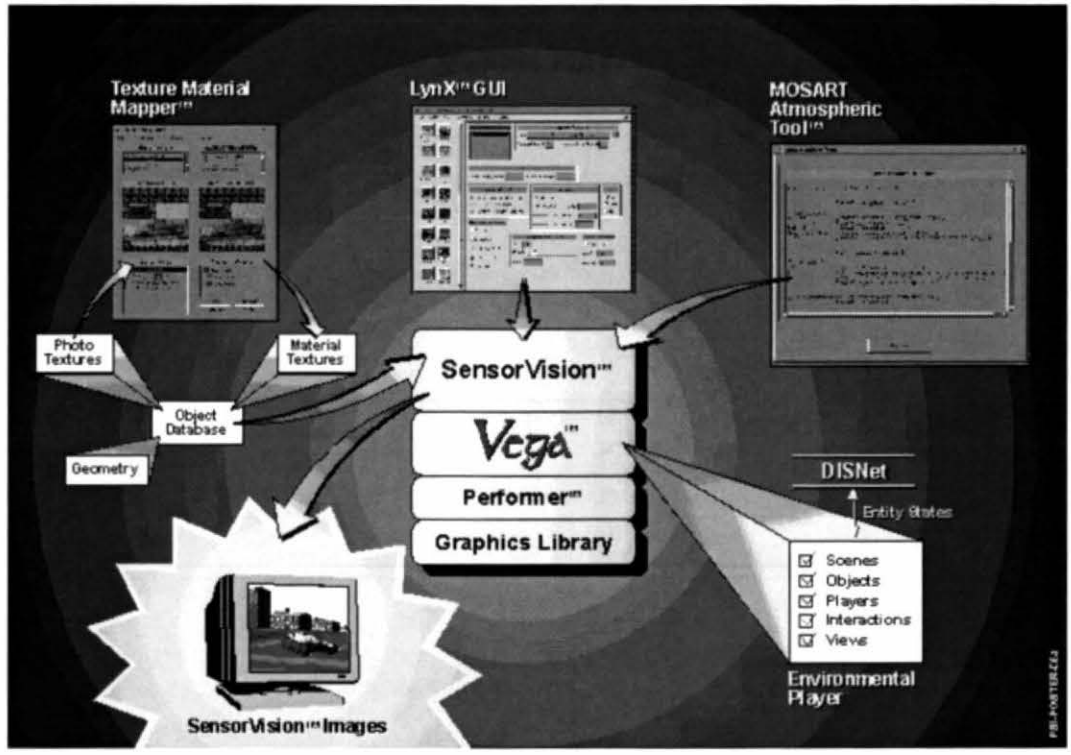


Figure 1. Production of SensorVision Imagery.

SensorVision operating with Vega generates both visual color "out-the-window" displays of a synthetic environment and sensor views of the same environment which are 100% correlated to the visual displays. SensorVision input is the same database used by Vega with two additions:

1. Textures of material codes. These are generated using the TMM tool to material code each Vega photographic texture, one at a time, from RGB colors into a linear combination of three materials, per texel.

2. An atmospheric and surface temperature database. These are generated using MAT for the latitude, longitude, date, and weather chosen by the user. A 24-hour time period is represented for each database. The atmospheric database is specific to the sensor spectral response function.

Application Definition Files (ADFs) can be generated using the Vega Lynx™ graphical user interface, or through Application Programming Interface (API). The Vega DIS module (which uses the VR-Link library) allows entity states to change dynamically, which includes position, orientation, and the surface temperatures of their parts (chosen during database construction). This state change information can be communicated over a network and read and processed by SensorVision.

2 1 SensorVision Features

SensorVision s features include

a Quantitative SensorVision evaluates a radiometric equation during its image visualization process yielding pixel radiance expressed in absolute radiometric units of watts/cm^2/sr The components of this equation are shown in Figure 2

b Textured in a Quantitative Environment Vega generates textured images but their color remains constant and is not quantitative SensorVision images are quantitative (polygon radiance is computed by trilinear interpolation on vertex radiance) and textured (polygon radiance is modified by the texture texel reflectance)

c Dynamic The SensorVision image visualization process can accommodate dynamic changes to the environment or the view perspective These include sun/moon position surface temperatures entity dynamic and temperature state and viewer altitude line of sight elevation and the azimuth angle between the line of sight and the position of the sun/moon

d Real Time SensorVision can be configured to compute images at the same speed as a Vega application (e g 30 Hz)

e Complete Environments The synthetic environment which SensorVision can process includes natural terrain backgrounds cultural objects mobile objects and all atmospheric states which are homogeneous in the horizontal dimension Treatment of non homogeneous atmospheric states are presently being developed In the SensorVision image visualization process the totality of the environment is represented by the positions of polygons the properties of the polygon vertices and their textures and the radiative environment

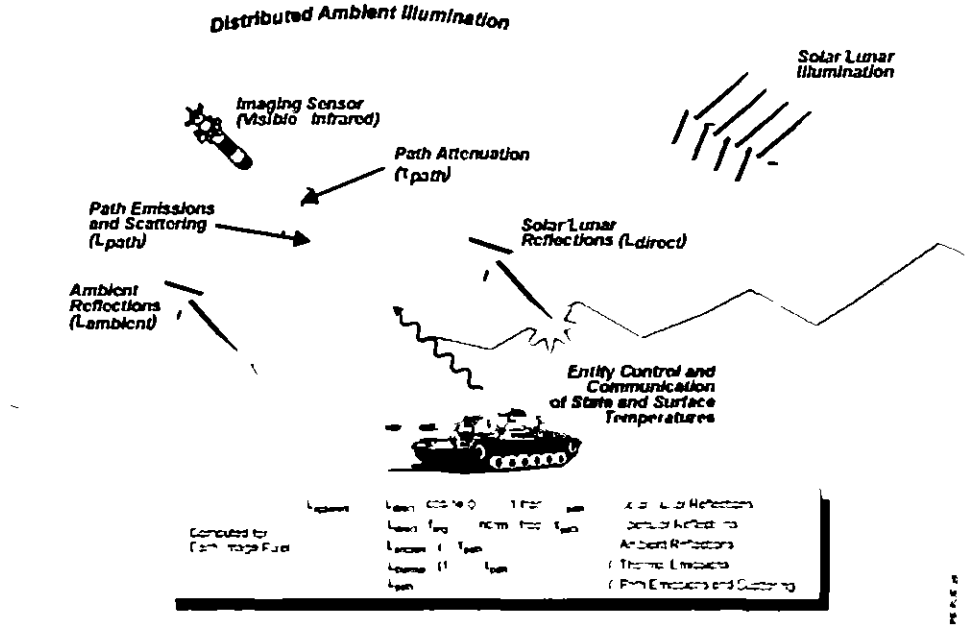


Figure 2 Quantitative Image Visualization in SensorVision

When configuring the environment the system positions all environmental polygons and textures specifies the vertex and texture properties positions all viewers and their windows and specifies time to configure the state of the environment and the perspective views These functions are done dynamically at each time step of the simulation

f Selectable Sensor Spectral Filters SensorVision can accommodate any sensor spectral filter from the visible through the infrared The present constraint is imposed by the material spectral reflectance database and is 0 42 to 14 0 μm

g DIS Compliant SensorVision accommodates the full Vega DIS module capability for distributed interactive simulation The DATA PDU is used to communicate dynamic changes to entity part temperatures

2 2 SensorVision Processing

The SensorVision system consists of four elements as shown in Figure 3 The primary executable is Vega + SensorVision which is an application built from the SensorVision library It is this executable which generates the image visualization It requires five inputs These are

1 The ADF which sets the simulation configuration and the initial state The ADF names the database input files and the directory paths to these files as well as the application executable

2 The Vega object flight and photo texture files

3 The TMM directory of material classified photo textures

4 The MAT directory of atmospheric quantities and material surface temperatures

5 The Materials database which contains the spectral reflectance data for all materials in the TMM classified textures

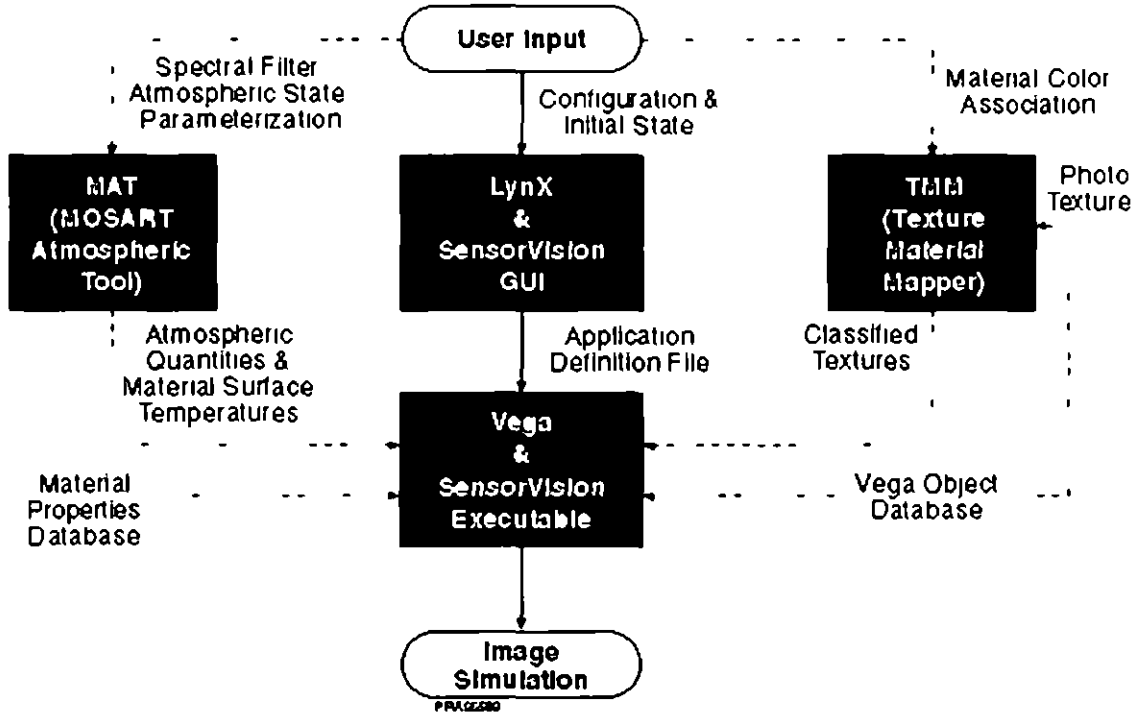


Figure 3 SensorVision System and Processing Flow

The inputs to MAT are all user selected and include the number and choices for atmospheric states the number

and wavelength response pairs for the spectral filters and the parameterizations for the material surface temperatures and the atmospheric quantities An example of a MAT input file is shown in Figure 4

```
1     { # of atmospheric states }

{ 1st state }
41 -112     { season  latitude (deg)  longitude (deg) }
1995 2 18   { season  year [1950,2050], month [1,12], day [1,31] }
Default     { season  model atmosphere name }
Mean Mean Mean { weather  temperature, humidity, wind }
Clear 1     { weather  cloud cover, cloud base altitude (km) [1 5] }
23    { weather  visibility range (km) (0,100] }

1     { # of spectral bands }

{ 1st band }
100       { spectral resolution (1/cm) }
2     { # of wavelength-response pairs [2,10000] }
3 1       { wavelength (microns), response [0,1] }
5 1       { wavelengths in increasing order }

{ parameterization for heat-transfer temperatures }
1     { # of materials }
ALL       { material names follow }
3     { # of surface altitudes [1,11] }
0 0 5 1     { altitudes (km) [0,10) }
5     { # of surface orientation azimuths [1,5] }
0 90 180 270 360 { azimuths (deg) [0,90,180 270,360] }
4     { # of surface orientation slopes [1,4] }
0 20 50 90  { slopes (deg) [0,180] }
25    { # of time-of-days [1,97] }
      { times (hours) [0,0 25,0 5,    23 75,24] }
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

{ parameterization for atmospheric quantities }
3     { # of line-of-sight ranges }
0 1 1 10 { ranges (km) [0 001,1000] }
2     { # of observer altitudes }
0 1 1     { altitudes (km) [0 002,1000] }
6     { # of line-of-sight elevations }
      { (scaled) elevations (deg) [-90,90] }
-90 -30 -5 5 30 90
6     { # of solar/lunar elevations }
      { (scaled) elevations (deg) [-90,90] }
-1 1 5 10 30 90
6     { # of solar/lunar-observer azimuths }
      { azimuths (deg) [0,180] }
0 90 140 170 178 180
```

Figure 4 MAT Example Input File

The inputs to TMM are the photo texture files These are material classified one by one by making associations between RGB colors in the photo texture and materials in the Material Property Database The materials in the SensorVision sample database consist of 95 soils vegetation construction composites and paints These are listed in Figure 5 Note that each paint may have either of four substrates asphalt, concrete wood or metal For each material values of diffuse spectral reflectance are given from 0 42 to 14 μm The user can expand this database by simply adding a material name and its associated tabular listing of wavelength and diffuse reflectance

SOILS (35)	VEGETATION (15)	CONSTRUCTION (16)	COMPOSITES (13)	PAINTS (16)
Lump Coal	Tundra	Roof Tar	Black Rubber	Jet Black
Water	Scrub	Black Asphalt	White Fabric	Snow White
Ice	Pine	Pebbled Asphalt	Grey Fabric	Off White
Fresh Snow	Broadleaf	Paved Concrete	Brown Plastic	Ash Grey
Old Snow	Lawn Grass	Block Concrete	Tan Felt	Battleship Grey
Road Gravel	Dry Grass	Red Brick	Beige Fabric	Blackish Brown
Limestone Rock	Scrub-Soil	Pine Wood	Cream Fiberglass	Dark Tan
Sandstone Rock	Grass Soil	Clear Glass	Gold Nylon	Light Tan
Varnished Sandstone	Grass-Scrub	Oxidized Aluminum	Olive Plastic	Sky Blue
Varnished Sand	Pine Scrub	Rusting Steel	Green Canvas	Olive Green
Black Sand	Broadleaf Scrub	Dark Titanium	Yellow Silicon	Kelly Green
White Sand	Broadleaf Pine	Metal Roof	Orange Epoxy	Mint Green
Lake Sand	Pine-Broadleaf	Urban Commercial	Red Nylon	Canary Yellow
Desert Sand	Scrub-Grass-Soil	80% Asphalt		Mustard Yellow
Beach Sand	Tree Bark	10% Vegetation		Bleached Yellow
Loamy Sand		10% Soil		Light Red
Sandy Loam		Urban Residential		
Silty Loam		45% Asphalt		
Silty Clay		10% Vegetation		
Clay Soil		5% Soil		
Loam Soil		Asphalt Shingles		
Compact Soil		Wood Siding		
Tilled Soil				
Wet Soil				
Wet Lakebed				
Dry Lakebed				
Wet Seabed				
Dry Seabed				
Salt Silt				
Silt Sand				
Sand-Soil				
Sandstone Soil				
Limestone Silt				
Limestone Silt Sand				
Limestone Silt Soil				

• Each paint may have a substrate of either asphalt, concrete wood, or metal

Figure 5 SensorVision Sample Materials Database

3 0 SENSORVISION RADIOMETRIC EQUATION

The radiometric equation evaluated by SensorVision during the image visualization process is

$$\begin{aligned} L_{observer} &= L_{direct} \cos\theta_i \, \rho \, \tau_{path} && \text{// solar/lunar reflections} \\ &+ L_{ambient} \, \rho \, \tau_{path} && \text{// skyshine reflections} \\ &+ L_{bb} \, (1 \quad \rho) \, \tau_{path} && \text{// thermal emissions} \\ &+ L_{path} && \text{// path emissions and scattering} \end{aligned} \qquad (1)$$

where

$L_{observer}$	=	radiance at observer	W/cm^2/sr
L_{direct}	=	solar/lunar radiance at surface	W/cm^2/sr
$L_{ambient}$	=	skyshine radiance at surface	W/cm^2/sr
L_{bb}	=	thermal (blackbody) radiance at surface	W/cm^2/ sr
L_{path}	=	atmospheric path radiance between surface and observer	W/cm^2/sr
$\cos\theta_i$	=	cosine of angle between surface normal and direction to sun/moon	dimensionless
ρ	=	surface diffuse reflectance	dimensionless
τ_{path}	=	atmospheric path transmission between surface and observer	dimensionless

The equation is grouped into four components These are solar/lunar reflections skyshine reflections thermal emissions and path emissions and scattering

3 1 Solar/l unar Reflections

This component is given by $I_{direct} \cos\theta_i \rho \tau_{path}$ where I_{direct} is computed through evaluation of the equation

$$L_{direct} = \frac{\frac{A_{sun/moon}}{\pi R_1^2} \int_{\lambda_1}^{\lambda_2} \phi(\lambda) L_{sun/moon}(\lambda) \tau_{R_1}(\lambda) \tau_{R_2}(\lambda) d\lambda}{\tau^*_{path}} \quad (2)$$

where

$A_{sun/moon}$	area of sun/moon
$L_{sun/moon}(\lambda)$	spectral radiance of sun/moon
$\tau_{R_1}(\lambda)$	spectral transmission of R_1 path
$\tau_{R_2}(\lambda)$	spectral transmission of R_2 path
$\phi(\lambda)$	sensor spectral response
$I_{R_1}(\lambda)$	spectral radiance of R_1 path
R_1	distance to sun/moon
λ	wavelength

A schematic of the direct reflection geometry is given in Figure 6

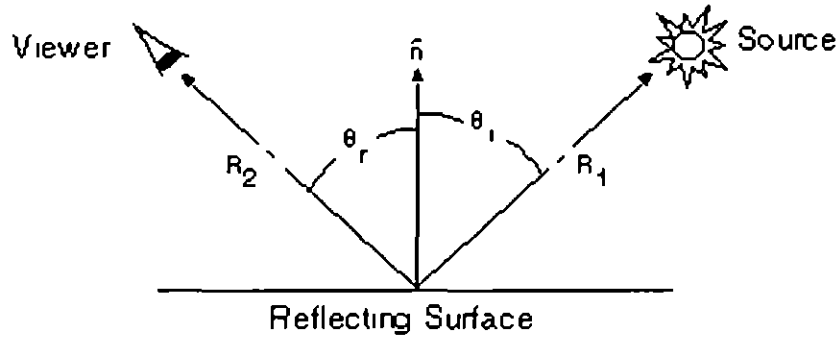


Figure 6 Schematic of Direct Solar/Lunar Reflection from Surface R_1 = distance to sun/moon R_2 = distance to viewer

τ^*_{path} is a transmission calibration factor for the path between the observer and the surface at the image center and is given by

$$\tau^*_{path} = \int_{\lambda_1}^{\lambda_2} \phi(\lambda) \tau_{R_2}(\lambda) d\lambda \quad (3)$$

MAT performs a rigorous evaluation of the numerator in Equation 2 However this is only done once for the line of sight at the image center An exponential extrapolation is performed to obtain the value of τ_{path} in Equation 1 for off center image pixels L_{direct} is exact at the image center but an approximation is made for other pixels The magnitude of the error in pixel radiance increases in proportion to the difference between the value of the pixel line of sight range between the observer and the surface and the value of the line of sight range at the image center Because of this SensorVision is most accurate for small fields of view

3 2 Skyshine Reflections

This component is given by $L_{ambient}\,\rho\tau_{path}$ where $L_{ambient}$ is computed through the evaluation of the equation

$$L_{ambient} = \frac{\frac{1}{\pi}\int_{\lambda_1}^{\lambda_2}\int_{0}^{2\pi}\int_{0}^{\pi/2}\phi(\lambda)\,L_{R_1}(\theta\ \phi\ \lambda)\,\tau_{R_2}(\lambda)\cos\theta\sin\theta\,d\theta\,d\phi\,d\lambda}{\tau^{*}_{path}} \tag{4}$$

where $\phi(\lambda)$ $\tau_{R2}(\lambda)$ and τ^{*}_{path} are as before and $L_{R1}(\theta\ \phi\ \lambda)$ is the spectral radiance of the sky for elevation angle θ and azimuth angle ϕ The integration is performed over the upper hemisphere MAT evaluates $L_{ambient}$ in a manner analogous to its evaluation of L_{direct}

3 3 Thermal Emissions

This component is given by $L_{bb}(1\ \ \rho)\,\tau_{path}$ where L_{bb} is given by

$$L_{bb} = \int_{\lambda_1}^{\lambda_2}\phi(\lambda)\,L(\lambda)\,d\lambda \tag{5}$$

and $L(\lambda)$ is given by Planck's equation

$$L(\lambda) = c_1/\lambda^5\left(e^{c_2/\lambda T} - 1\right)^{1} \tag{6}$$

where	c_1	=	$2\pi c^2h = 3\ 7413\times10^{-12}$	$W\ cm^2$
	c_2	=	$hc/k = 1\ 4388$	cm K
	h	=	$6\ 6252\times10^{-34}$	$W\ sec^2$
	k	=	$1\ 38042\times10^{-23}$	W sec/K
	c	=	$2\ 99793\times10^{10}$	cm/sec
	T	=	temperature	K

3 4 Path Emissions and Scattering

This component is given by

$$L_{path} = \int_{\lambda_1}^{\lambda_2}\phi(\lambda)\,L_{R_2}(\lambda)\,d\lambda \tag{7}$$

where $L_{R_2}(\lambda)$ spectral radiance of R_2 path

3 5 MAT Evaluations

Within MAT the terms L_{direct}, $L_{ambient}$, L_{bb} τ_{path} and L_{path} are evaluated by MOSART (Reference 5) for a range of values of the controlling parameters This results in a multivariate database for each quantity which is used to evaluate the respective terms of the radiance equation during run time These parameterizations for MAT are

Sun/Moon Radiance at Surface (L_{direct}) and Skyshine Radiance at Surface ($L_{ambient}$)

L_{direct} and $L_{ambient}$ are computed for user selected values of

1 Line of sight range between the observer and the surface
2 Observer altitude
3 Line of sight elevation angle
4 Direct source elevation angle

During run time values of line-of sight range observer altitude line of sight elevation angle and direct source elevation angle are computed for each time step for the image center and then L_{direct} and $L_{ambient}$ are computed using linear interpolation on this multivariate database

Thermal Blackbody Radiance at Surface (L_{bb})

In band values of blackbody radiance are computed as a function of temperature over the range of possible surface temperatures

MAT computes surface temperatures for all user specified materials as a function of user selected values of 1) surface altitude 2) surface orientation azimuth angle relative to the north 3) surface orientation slope relative to the horizontal and 4) time of day

MAT computed temperatures are applicable to backgrounds only and should not be used for operated or mobile entities such as missiles airplanes and tanks For these objects polygon vertex temperatures are supplied to SensorVision through API

During run time SensorVision assigns a temperature per polygon vertex according to the vertex material its normal vector and the current time Blackbody radiance is computed for these temperatures and pixel radiance is computed using trilinear interpolation from the vertex radiances

Atmospheric Path Transmission (τ_{path}) and Path Radiance (L_{path})

In band values are computed for user selected values of

1 Line of sight range between the observer and the surface
2 Observer altitude
3 Line of sight elevation angle
4 Direct source elevation angle
5 Direct source and observer azimuth angle

During run time the values of τ_{path} and L_{path} are computed for the line of sight at the image center using linear interpolation on this multivariate database This is done per image frame The values of τ_{path} and L_{path} used in the radiance equation for each image pixel are computed using exponential extrapolation according to pixel line of-sight range between the observer and the surface

4 0 SENSORVISION USE IN HARDWARE IN THE LOOP SIMULATIONS

SensorVision provides the scene image input needed for a real time hardware in the loop sensor simulation The following issues are relevant for this type of simulation application

1 Accuracy of quantitative calculation The graphics hardware does the calculations at 32 bit precision However the results are truncated to the frame buffer resolution which is at most 12 bits for RealityEngine2 and at most 16 bits for InfiniteReality Also the texture values (in-band reflectances) come in at 4 8 12 or 16 bits precision (user selectable)

2 Framing rate limit The maximum framing rate is limited by the scene database complexity and output frame size RealityEngine2 draws ~2 000 000 triangles/second and fills ~360 000 000 pixels/second while InfiniteReality draws ~10 000 000 polygons/second and fills ~800 000 000 pixels/second

3 Full scene anti aliasing Use of anti aliasing at scene generation time reduces the frame size needed at the sensor Not using anti aliasing the sensor resolution should be over sampled by a factor of four to compensate for pixel sampling artifacts Since the graphics hardware is capable of 4 8 or 16 point pixel sampling (user selectable) the frame size requirements are reduced

4 Texture MIP mapping of sub-pixel features To correctly account for the radiance of target hot spots or other sub-pixel features objects can be rendered at a higher resolution during a separate draw pass copied from the frame buffer into texture memory and used at regular resolution during the full scene draw pass The hardware texture MIP mapping function correctly integrates the higher resolution texels down to lower pixel resolution [This process is currently in development]

5 Latency/Framing rate trade-off The framing rate can be improved by up to a factor of three by using multi stage pipeline modes but at the cost of possibly introducing additional latency Switching between single and multiple stage pipeline modes is easily accomplished

6 Concurrent modeling The hardware multi processor architecture allows many application models to be run simultaneously with the image generation For example target thermal models running on separate processors can update vertex temperatures asynchronously to the framing loop (and with essentially no latency)

5.0 SUMMARY

This paper has given an overview of a COTS software system capable of providing sensor image input to hardware in-the loop simulators The features of the software and the issues of its use in hardware in the loop simulators have also been stated SensorVision and its associated software libraries are very capable in their present form However SensorVision capabilities are continuously being expanded and improved to meet the expanding needs of simulators and to take advantage of the rapid growth in computer technology

REFERENCES

1 MaK Technologies Inc "VR-Link The Virtual Reality Networking Toolkit " Release 2 4 1 August 1995

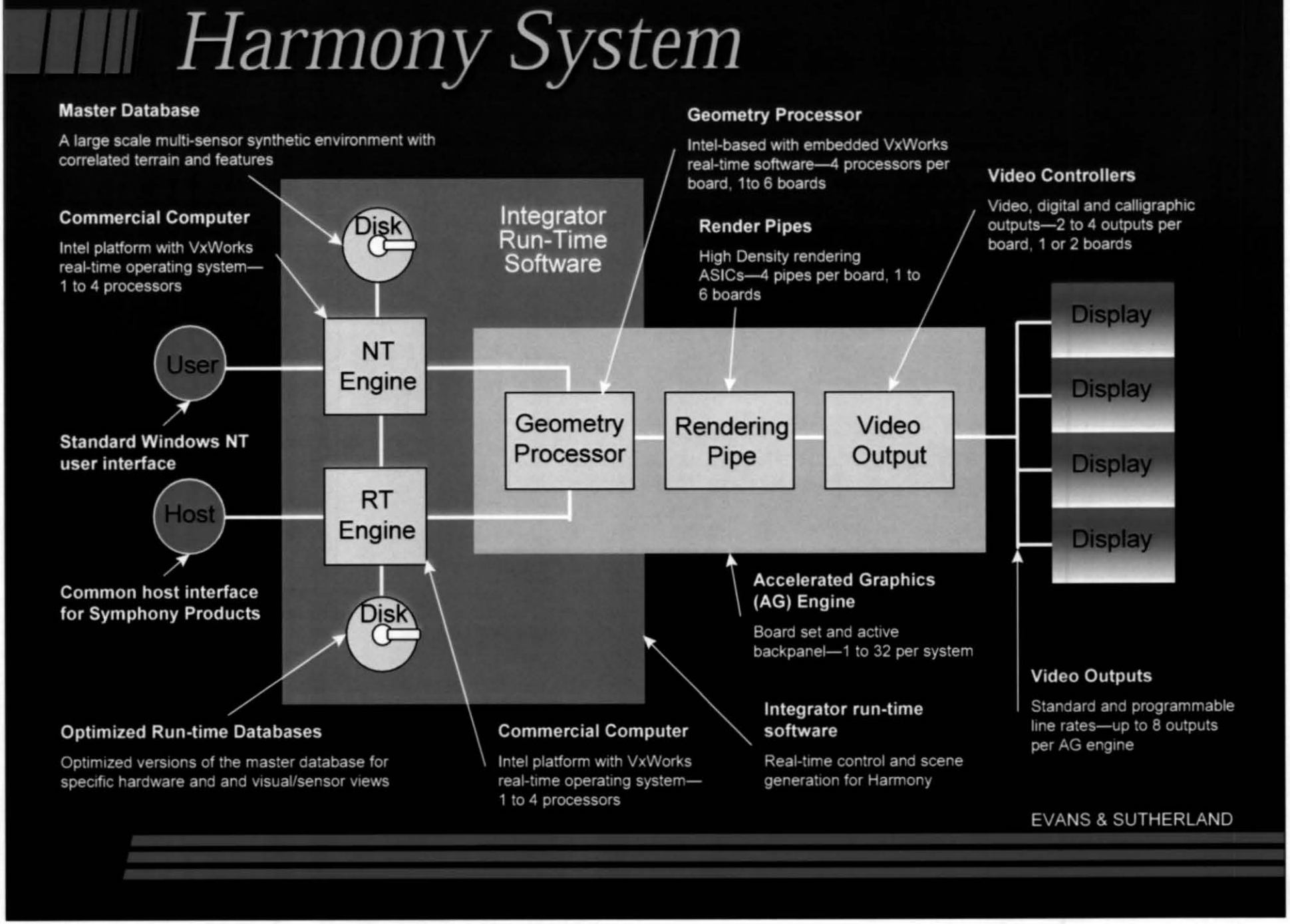
2 Paradigm Simulation, Inc "Vega Technical Overview " 1995

3 Paradigm Simulation, Inc "Vega Programmer's Guide " Version 2 0 1996

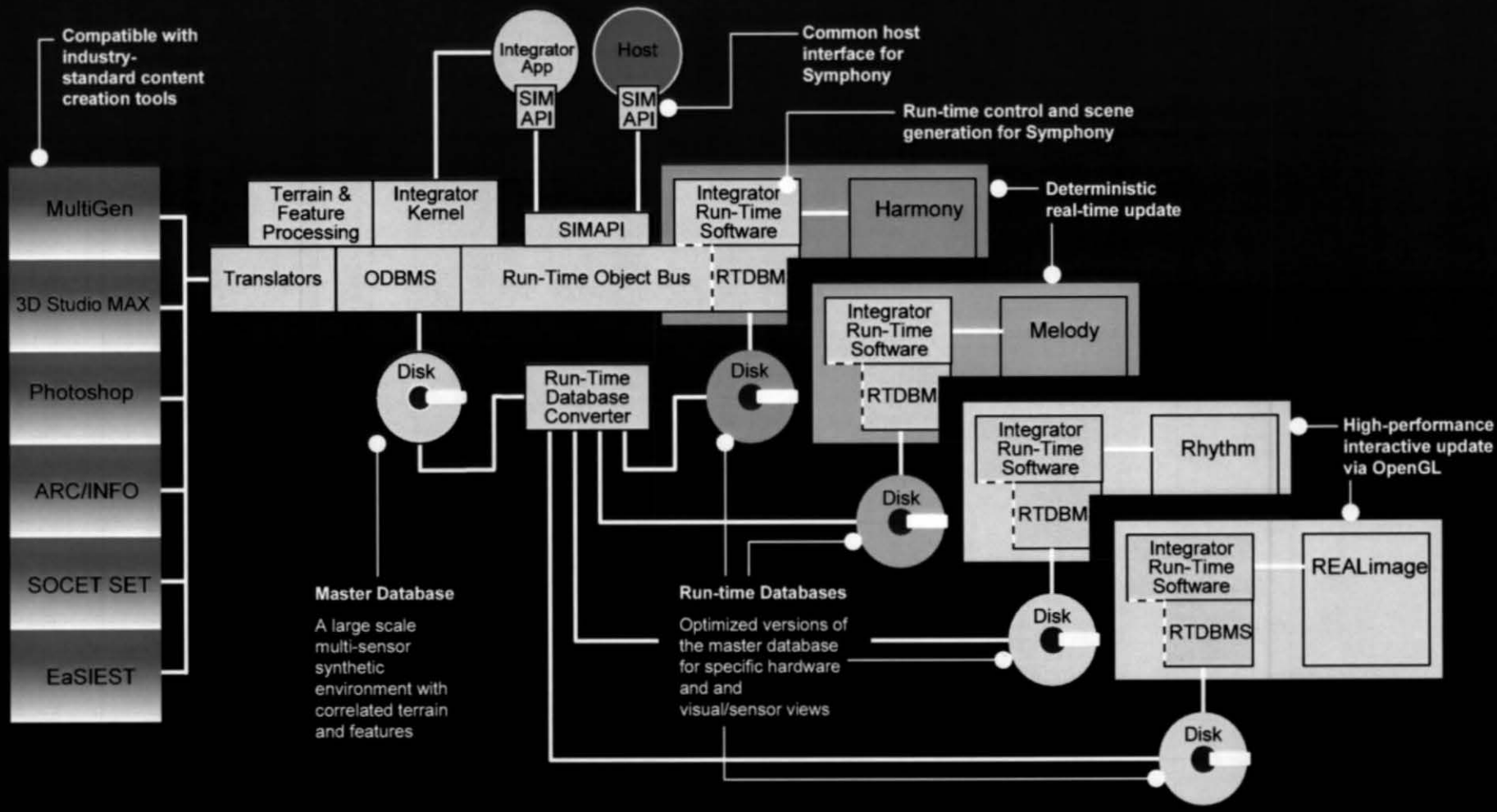
4 Paradigm Simulation Inc "LynX User's Guide " Version 2 0 1996

5 Photon Research Associates Inc "Moderate Spectral Atmospheric Radiance and Transmittance Program (MOSART) " Version 1 40 August 1995

Harmony System



Integrator Architecture and Dataflow



EVANS & SUTHERLAND

IRIS Performer™ 2.2

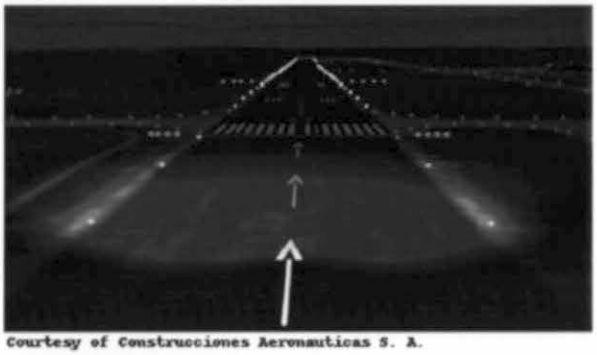
©1997 Delphi International

Courtesy of IVEX

©1997 Paradigm Simulation Inc.

©1997 SWRI

©1997 Boston Dynamics Inc.



Courtesy of Construcciones Aeronauticas S. A.

Courtesy of Construcciones Aeronauticas S. A.

IRIS Performer2.2 Image Credits

Delphi International – "Yosemite"
The Yosemite picture was created by Delphi International with Yosemite image data courtesy of the National Park Service The rendering of this image used Cliptexture to drape 5m image data onto an Active Surface Definition terrain with 5m elevation data Delphi International is a global provider of high-end data visualization software and services
Phone (650) 548-9696
Email delphi@delphi-intl com
http //www webville com/oak/delphi

IVEX – "Seattle-Tacoma International Airport"
IVEX provides real-time visual systems for commercial and military flight simulation Superior integrated image generators database scenarios and complete system integration services using Clipmapping ASD and Calligraphic lights
Phone (770) 564-1148
http //www ivex3d com

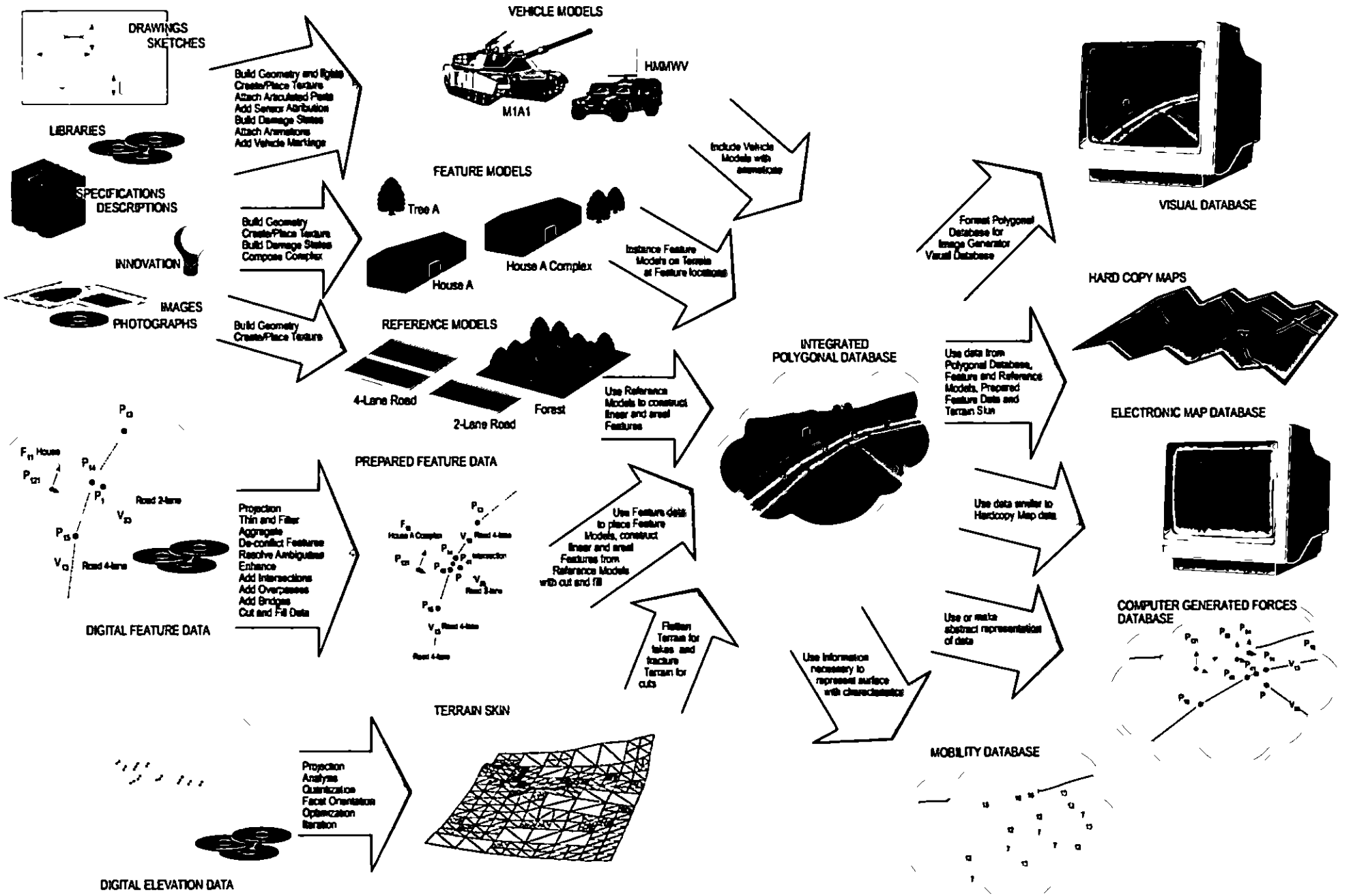
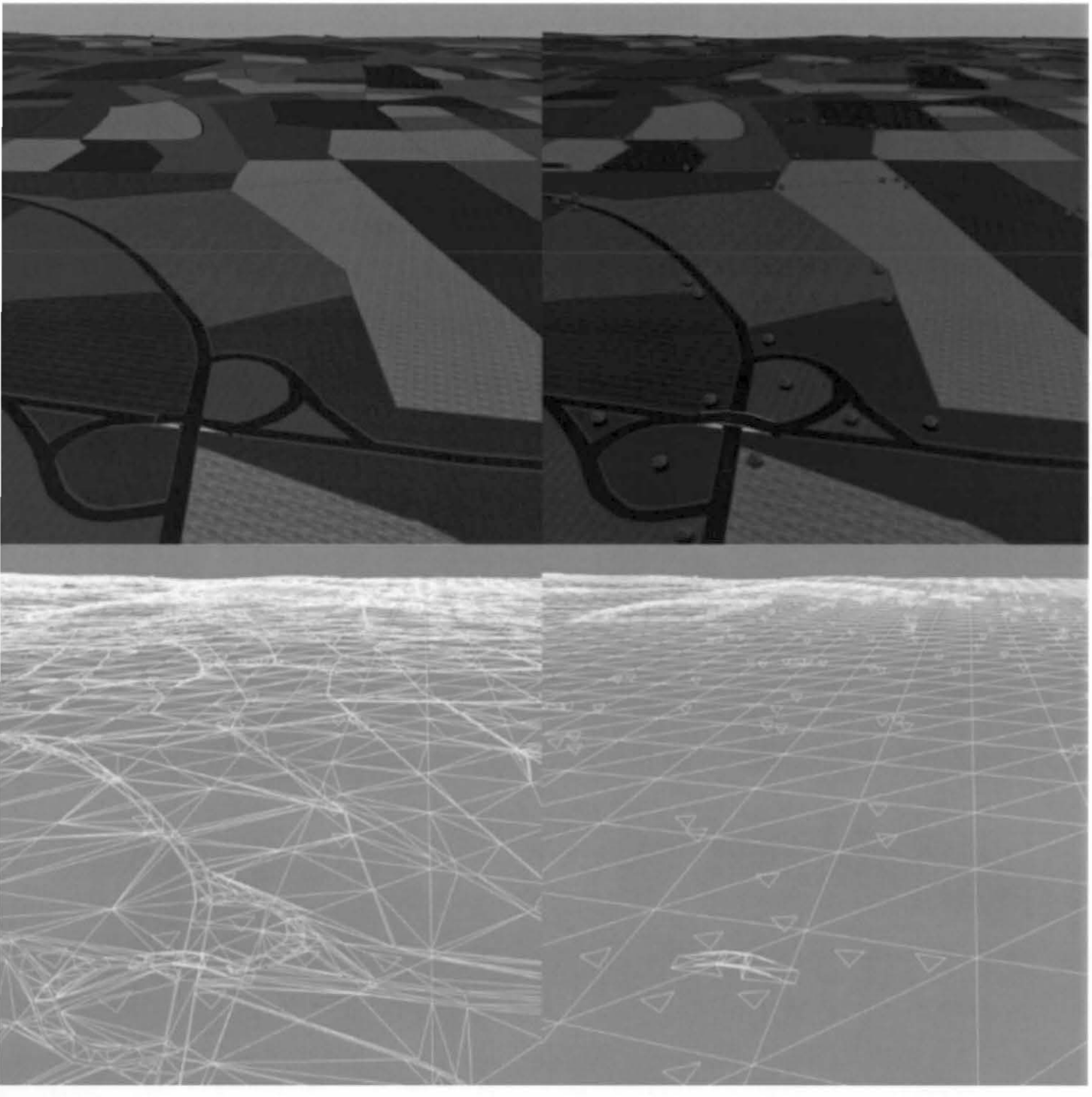
Paradigm Simulation Inc – "Ocean and Marine Effects Simulation"
Vega Marine is Performer based visual simulation software for the development of maritime applications This module one of many within the Vega Development system generates a real-time dynamic ocean modeled as a textured surface with wave heights and periods corresponding to sea states Vega Marine provides special effects essential to realistic maritime simulation These effects include wakes wind effects on water constant tension and constant length lines moored buoys depth (bathymetry) effects foam flotsam surf horizon glow and glare from the sun
Phone (972) 960-2301
Email marketing@paradigmsim com
http //www paradigmsim com

Southwest Research Institute – "Clouds"
Southwest Research Institute recieved an R&D 100 award for development of the patented Weather Environment Simulation Technology (WEST[TM]) process WEST[TM] enables volumetric rendering of real world weather conditions through the use of dynamically shaped billboards The process assures real-time performance by prioritizing weather elements in the field of view
Phone (210) 522-5001
Email BMontag@SwRI org
http //www swri org

Boston Dynamics Inc – "DI-Guy[TM]"
DI-Guy[TM] realistic humans for virtual environments produced by Boston Dynamics Inc Using run-time morphing for smoothly animated and interactive figures
Phone (617) 868-5600
http //www bdi com

CASA – "Night Image"
Image courtesy of Construcciones Aeronauticas S A Madrid Spain Aircraft 60 Hz Visual system with advanced visual effects including runway aeronautical model lighting Landing lobes implementation affects terrain following and cultural features illumination
Phone ++34/1/6242444

CASA – "Day Image"
Image courtesy of Construcciones Aeronauticas S A Madrid Spain Aircraft 60Hz visual system with fully geospecific RT database paging and advanced weather model


DRAWINGS
SKETCHES
LIBRARIES
SPECIFICATIONS
DESCRIPTIONS
INNOVATION
IMAGES
PHOTOGRAPHS
Build Geometry and Splats
Create/Place Texture
Attach Articulated Parts
Add Sensor Attributes
Build Damage States
Attach Animations
Add Vehicle Markings
Build Geometry
Create/Place Texture
Build Damage States
Compose Complex
Build Geometry
Create/Place Texture
VEHICLE MODELS
M1A1
HMMWV
FEATURE MODELS
Tree A
House A
House A Complex
REFERENCE MODELS
4-Lane Road
2-Lane Road
Forest
Include Vehicle Models with orientation
Instance Feature Models on Terrain at Feature locations
Use Reference Models to construct linear and areal Features
INTEGRATED POLYGONAL DATABASE
Format Polygonal Database for Image Generator Visual Database
VISUAL DATABASE
HARD COPY MAPS
Use data from Polygonal Database, Feature and Reference Models, Prepared Feature Data and Terrain Skin
ELECTRONIC MAP DATABASE
Use data similar to Hardcopy Map data
COMPUTER GENERATED FORCES DATABASE
Use or make abstract representation of data
MOBILITY DATABASE
Use information necessary to represent surface with characteristics
Flatten Terrain for lakes and fracture Terrain for cuts
Use Feature data to place Feature Models, construct linear and areal Features from Reference Models with cut and fill
PREPARED FEATURE DATA
Road 2-lane
Road 4-lane
Projection
Thin and Filter
Aggregate
De-conflict Features
Resolve Ambiguities
Enhance
Add Intersections
Add Overpasses
Add Bridges
Cut and Fill Data
DIGITAL FEATURE DATA
TERRAIN SKIN
Projection
Analysis
Quantization
Facet Orientation
Optimization
Iteration
DIGITAL ELEVATION DATA