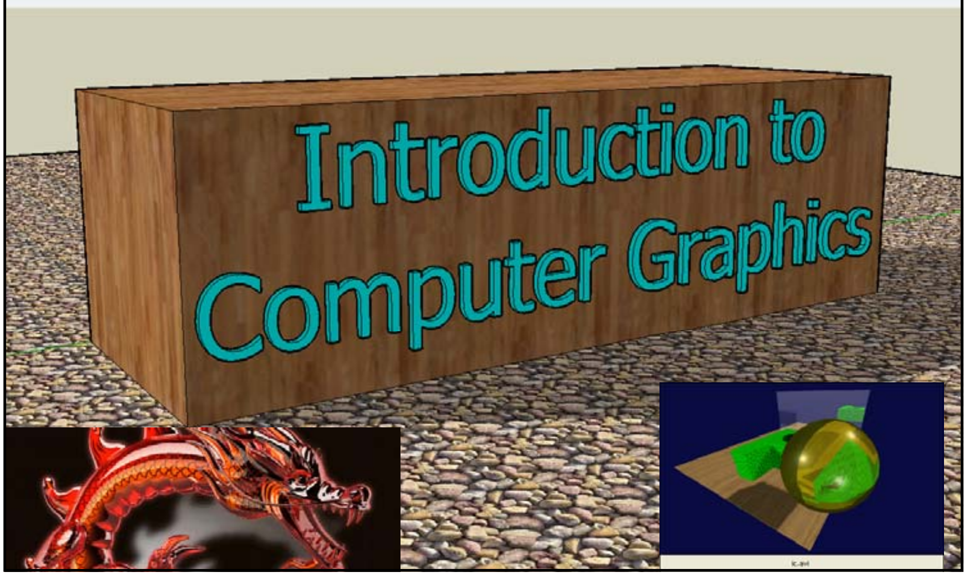
Mike Bailey
Oregon State University
mjb@cs.oregonstate.edu

Steve Cunningham
Brown/Cunningham Associates
rsc@cs.csustan.edu



Mike Bailey

- Professor of Computer Science, Oregon State University
- Has worked at Sandia Labs, Purdue University, Megatek, San Diego Supercomputer Center (UC San Diego), and OSU
- Has taught over 4,000 students in his classes
- mjb@cs.oregonstate.edu

Steve Cunningham

- Retired Professor of Computer Science, California State University Stanislaus
- Has served as chair of both the SIGGRAPH Education Board and the Eurographics Education Board
- Has written 7 books on computer graphics topics
- rsc@cs.csustan.edu

Course Goals

- Provide a background for papers, panels, and other courses
- Create a common understanding of computer graphics vocabulary
- Help appreciate the images you will see
- Get more from the Exhibition
- Provide pointers for further study

Specific Topics

- The Graphics Process
- Graphics Hardware
- Modeling
- Rendering
- GPU Shaders
- Finding More Information

Schedule

0:00 Welcome and Overview
0:10 The Graphics Process
0:40 Graphics Hardware
1:10 Modeling

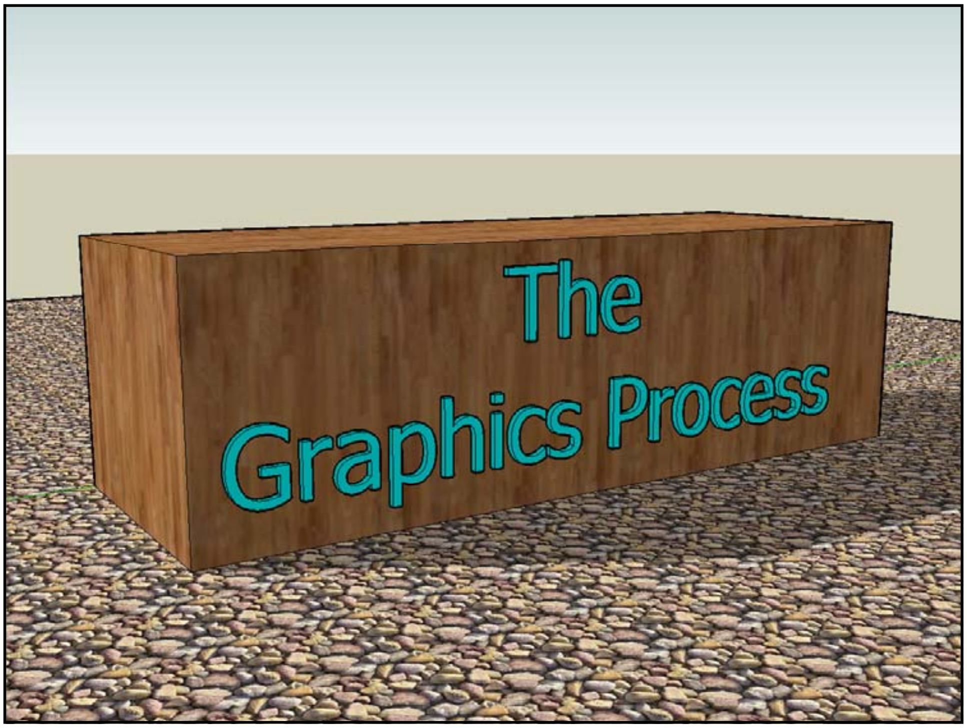
1:45 Break

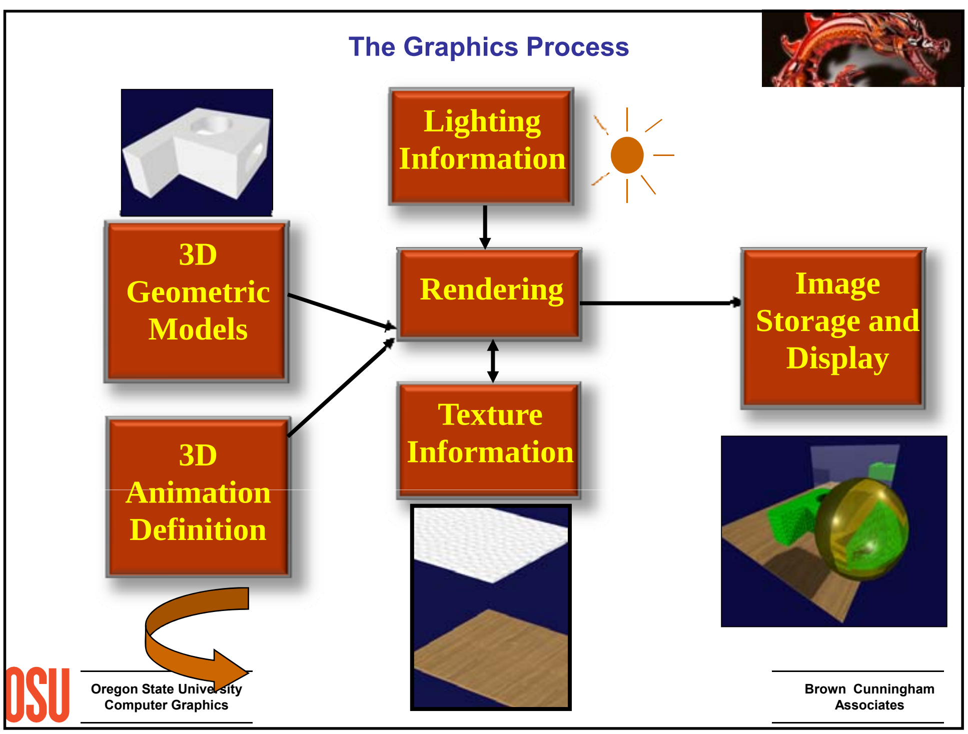
2:00 Maybe our vision isn't as good as we think it is ☺
2:15 Rendering
2:30 GPU Shaders
2:50 Finding Additional Information
3:10 Discussion and Questions

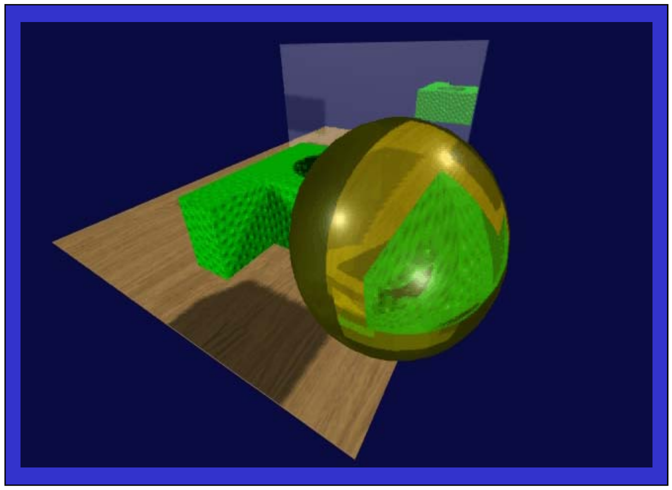
3:30 Finish

Oregon State University
Computer Graphics

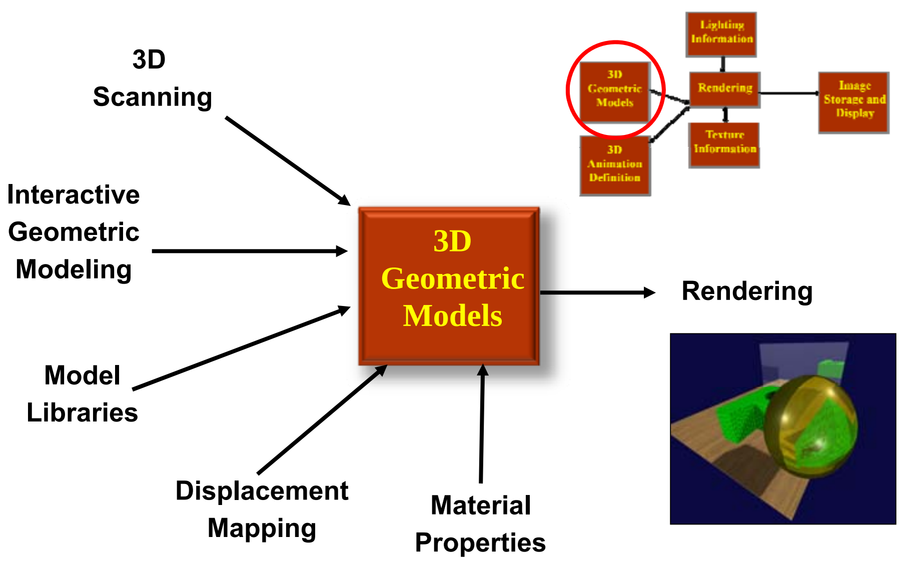
Brown Cunningham
Associates

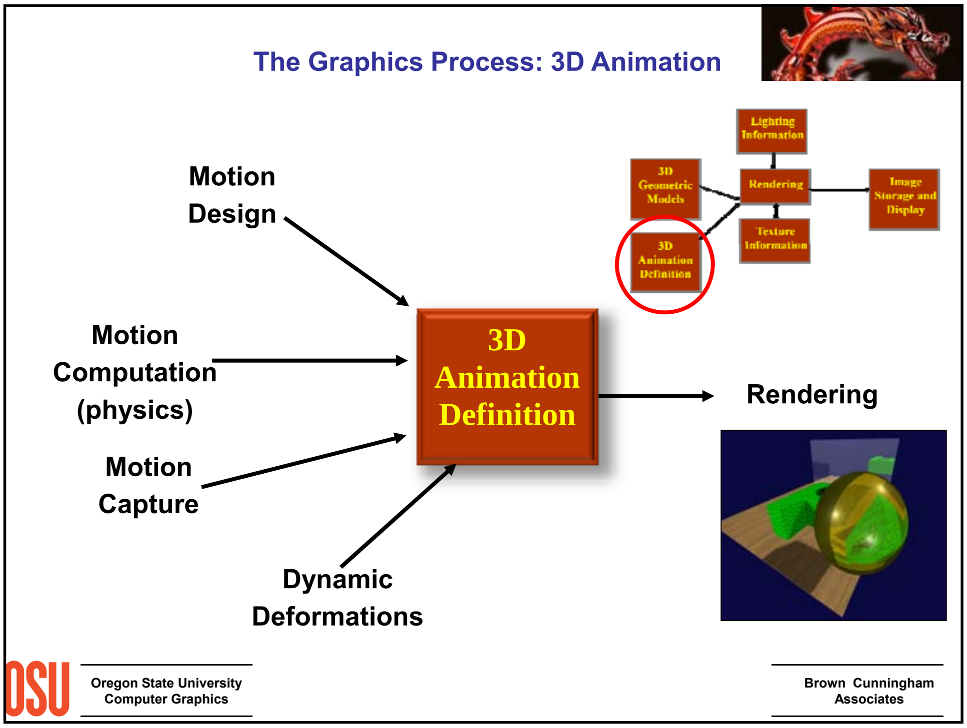

The
Graphics Process

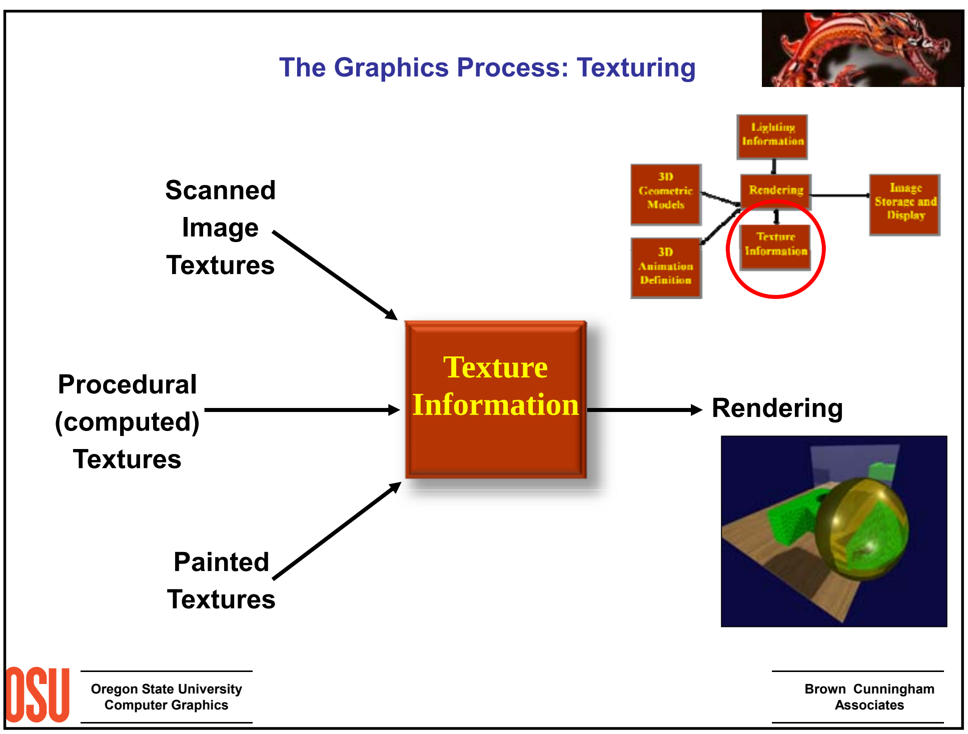

The Graphics Process
Lighting Information
3D Geometric Models
Rendering
Image Storage and Display
3D Animation Definition
Texture Information
OSU
Oregon State University
Computer Graphics
Brown Cunningham
Associates



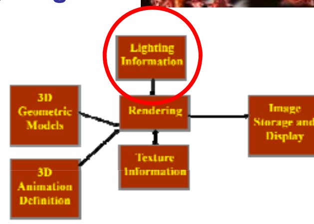
The Graphics Process: Geometric Modeling




The Graphics Process: 3D Animation
Lighting Information
3D Geometric Models
Rendering
Image Storage and Display
Texture Information
3D Animation Definition
Motion Design
Motion Computation (physics)
Motion Capture
Dynamic Deformations
3D Animation Definition
Rendering
Oregon State University Computer Graphics
Brown Cunningham Associates


The Graphics Process: Texturing
Lighting Information
3D Geometric Models
Rendering
Image Storage and Display
Texture Information
3D Animation Definition
Scanned Image Textures
Procedural (computed) Textures
Painted Textures
Texture Information
Rendering
Oregon State University Computer Graphics
Brown Cunningham Associates

The Graphics Process: Lighting



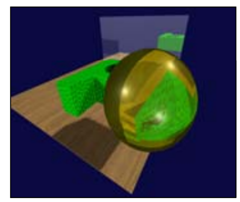
Lighting Types (point, directional, spot) → **Lighting Information**

Light Positions → **Lighting Information**

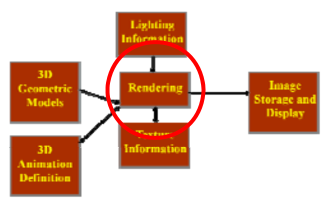
Light Colors → **Lighting Information**

Light Intensities → **Lighting Information**

Lighting Information → Rendering



The Graphics Process: Rendering



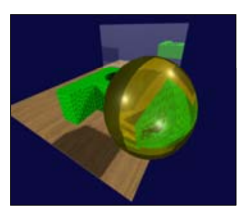
3D Geometric Models → **Rendering**

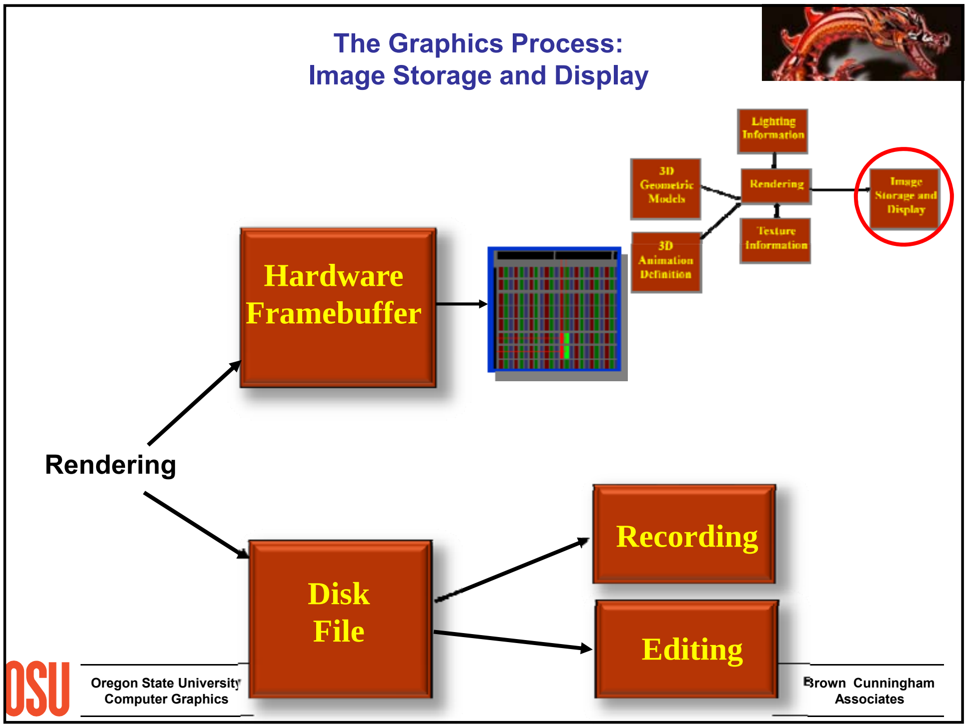
Lighting Information → **Rendering**

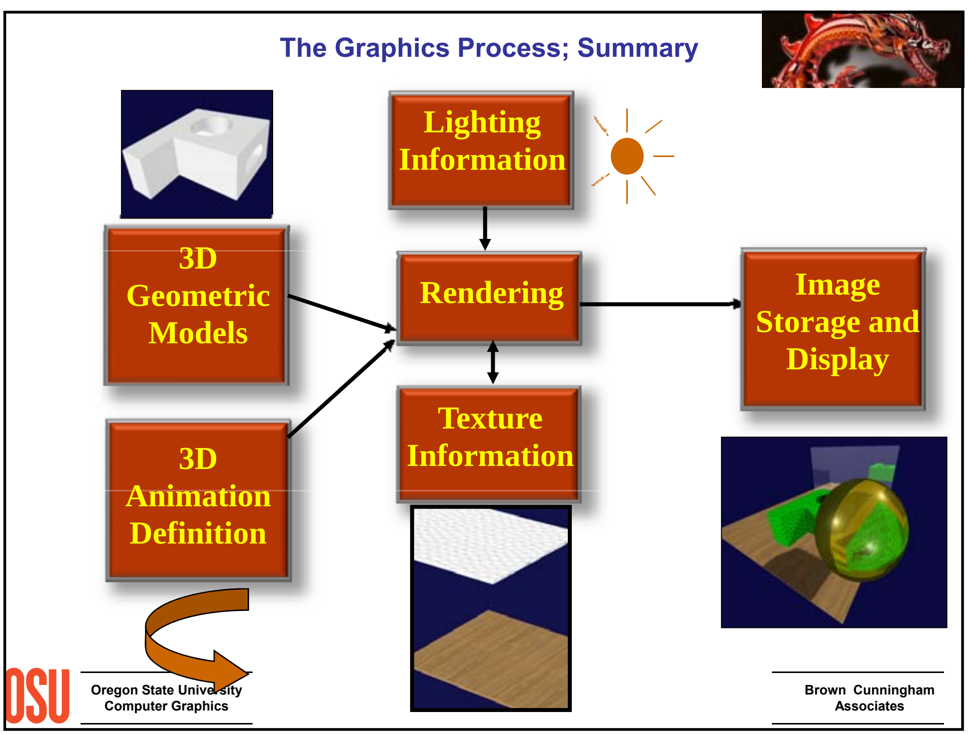
3D Animation Definition → **Rendering**

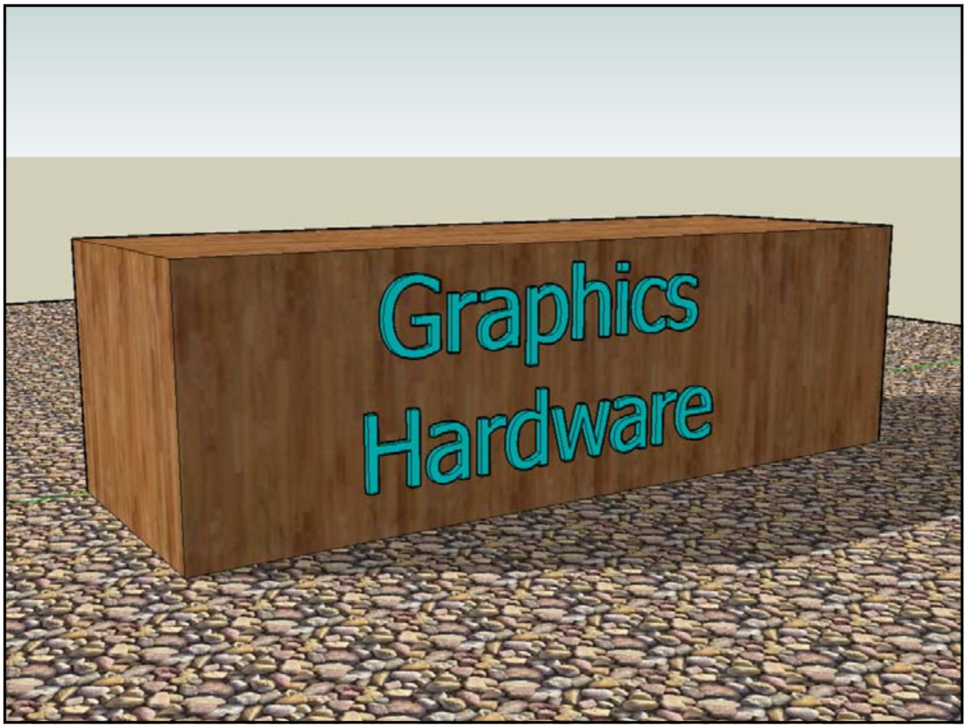
Texture Information ↔ **Rendering**

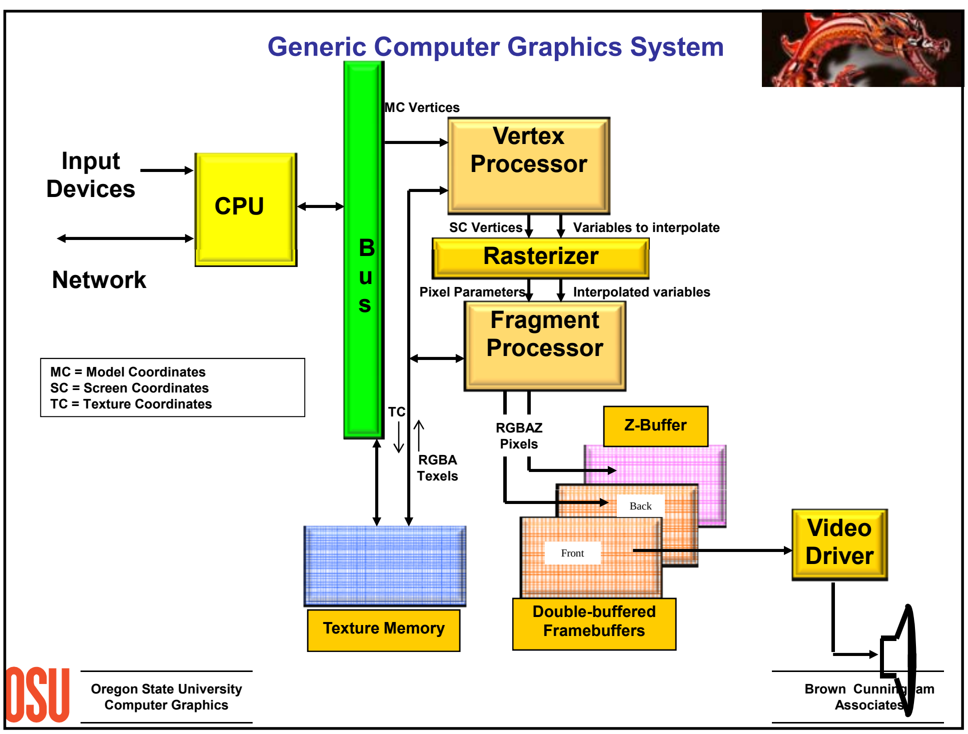
Rendering → Image Storage and Display



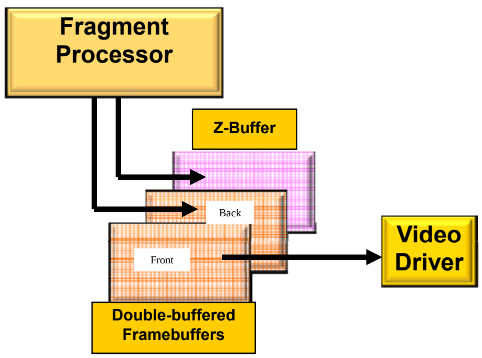

The Graphics Process:
Image Storage and Display
Lighting Information
3D Geometric Models
Rendering
Image Storage and Display
Texture Information
3D Animation Definition
Hardware Framebuffer
Rendering
Recording
Disk File
Editing
OSU
Oregon State University
Computer Graphics
Brown Cunningham
Associates

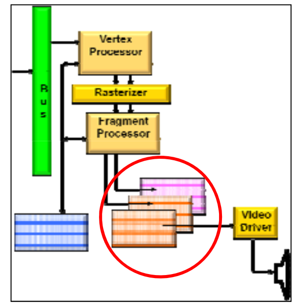

The Graphics Process; Summary
Lighting Information
3D Geometric Models
Rendering
Image Storage and Display
3D Animation Definition
Texture Information
OSU
Oregon State University
Computer Graphics
Brown Cunningham
Associates


Graphics
Hardware


Generic Computer Graphics System
Input
Devices
Network
CPU
B
u
s
MC Vertices
Vertex
Processor
SC Vertices
Variables to interpolate
Rasterizer
Pixel Parameters
Interpolated variables
Fragment
Processor
MC = Model Coordinates
SC = Screen Coordinates
TC = Texture Coordinates
TC
RGBA
Texels
RGBAZ
Pixels
Z-Buffer
Back
Front
Video
Driver
Texture Memory
Double-buffered
Framebuffers
OSU
Oregon State University
Computer Graphics
Brown Cunningham
Associates

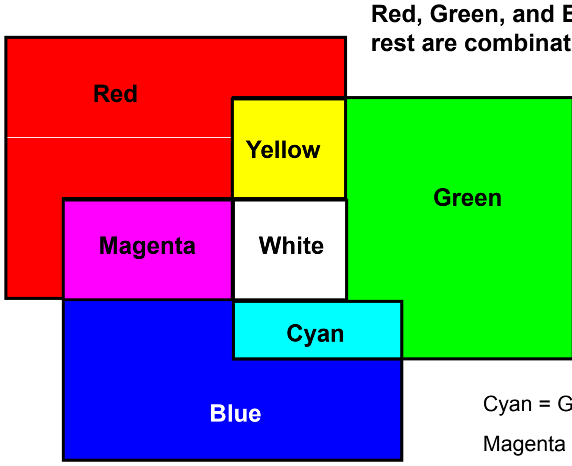
The Framebuffer





The Framebuffer Uses Additive Colors (RGB)

Red, Green, and Blue are provided. The rest are combinations of those three.



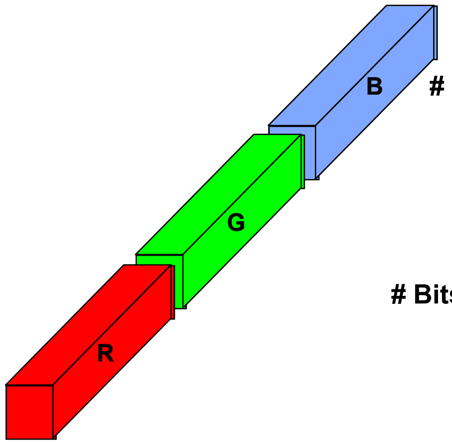
Cyan = Green + Blue

Magenta = Red + Blue

Yellow = Red + Green

White = Red + Green + Blue

The Framebuffer:
Integer Color Storage

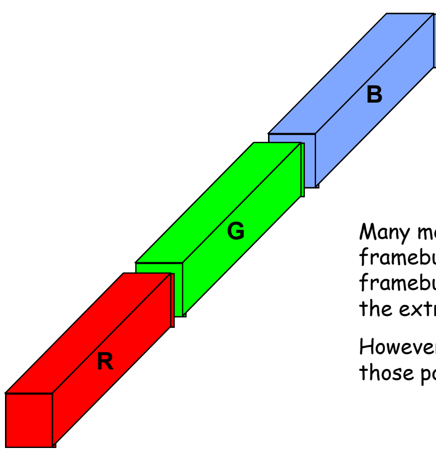


# Bits/color	# Intensities per color
8	$2^8 = 256$
10	$2^{10} = 1024$
12	$2^{12} = 4096$

# Bits/pixel	Total colors:
24	2^{24} = 16.7 M
30	2^{30} = 1 B
36	2^{36} = 69 B

The Framebuffer:
Floating Point Color Storage

- *16- or 32-bit floating point for each color component*



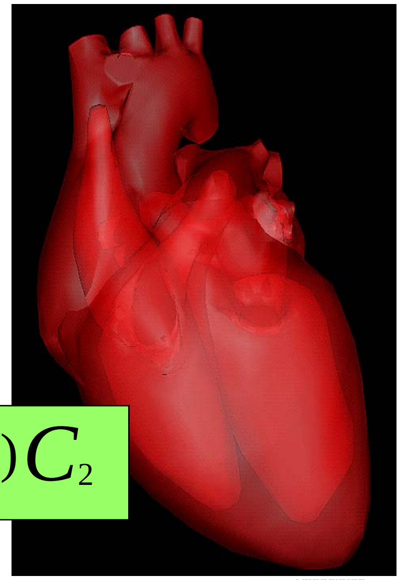
Why so much?

Many modern algorithms do arithmetic on the framebuffer color components, or treat the framebuffer color components as data. They need the extra precision during the arithmetic.

However, the display system cannot display all of those possible colors.

The Framebuffer

- ***Alpha*** values
  - Transparency per pixel
    α = 0. is invisible
    α = 1. is opaque
  - Represented in 8-32 bits
    (integer or floating point)
  - Alpha blending equation:

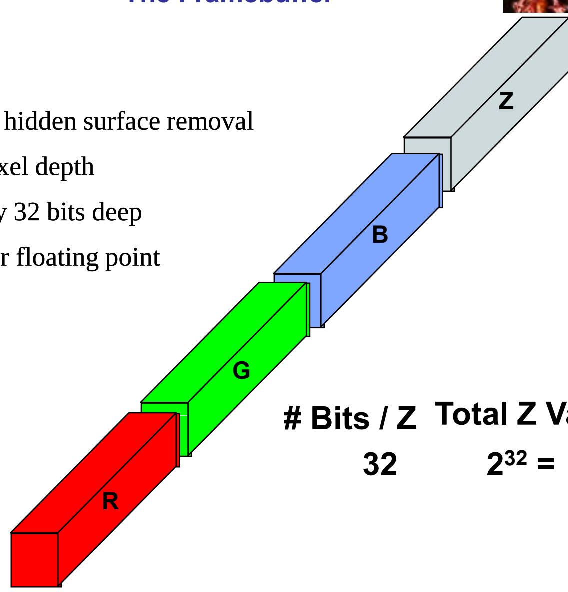


$$Color = \alpha C_1 + (1 - \alpha) C_2$$

$0.0 \leq \alpha \leq 1.0$

The Framebuffer

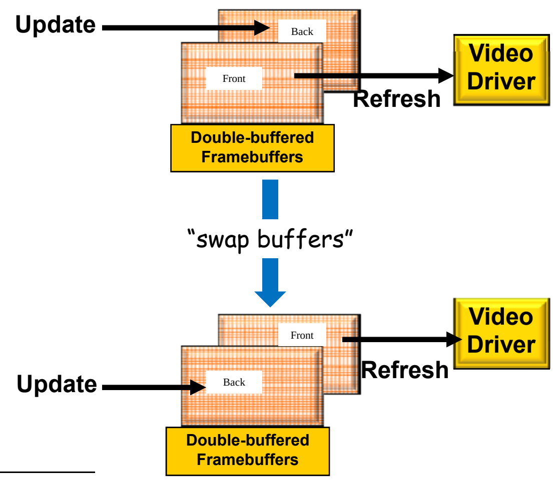
- ***Z-buffer***
  - Used for hidden surface removal
  - Holds pixel depth
  - Typically 32 bits deep
  - Integer or floating point



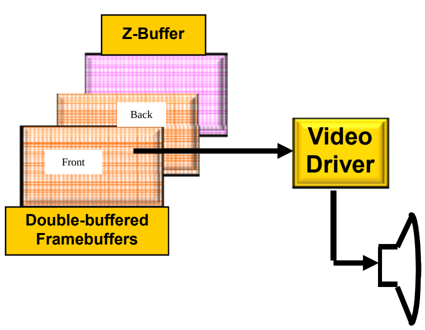
# Bits / Z	Total Z Values:
32	2^{32} = 4 B

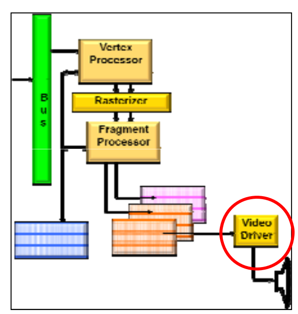
The Framebuffer

Double-buffering: Don't let the viewer see *any* of the scene until the entire scene is drawn



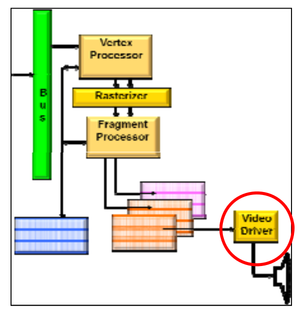
The Video Driver





The Video Driver

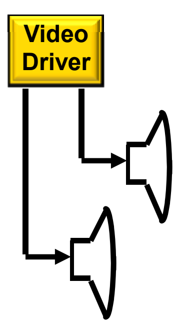
- N ***refreshes*/second** (N is usually between 50 and 100)
- Framebuffer contains the R,G,B that define the color at each pixel
- Cursor
  - Appearance is stored near the video driver in a “mini-framebuffer”
  - x,y is given by the CPU
- Video input

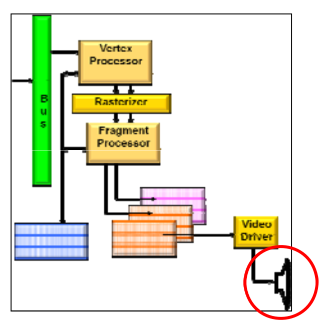


Oregon State University
Computer Graphics

Brown Cunningham
Associates

The Computer Graphics Monitor(s)



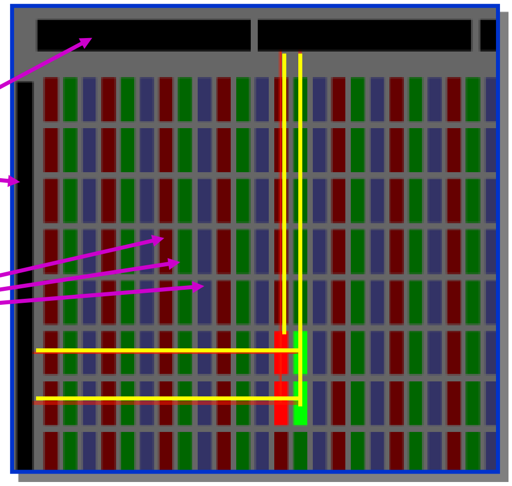


Oregon State University
Computer Graphics

Brown Cunningham
Associates

Displaying Color on a Computer Graphics LCD Monitor

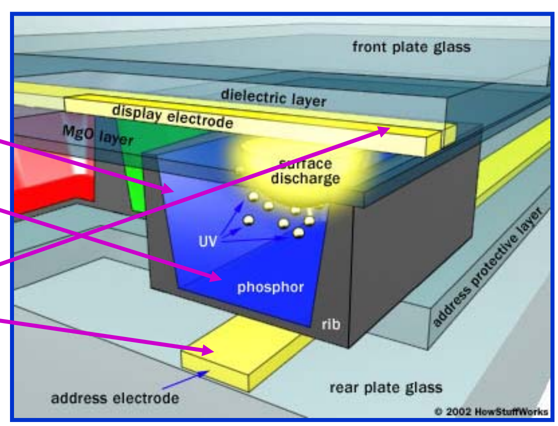
- Grid of electrodes
- Color filters



Source: http://electronics.howstuffworks.com

Displaying Color on a Plasma Monitor

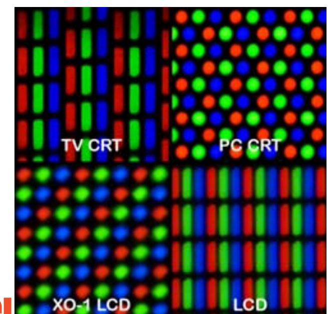
- **Gas cell**
- **Phosphor**
- **Grid of electrodes**



http://electronics.howstuffworks.com

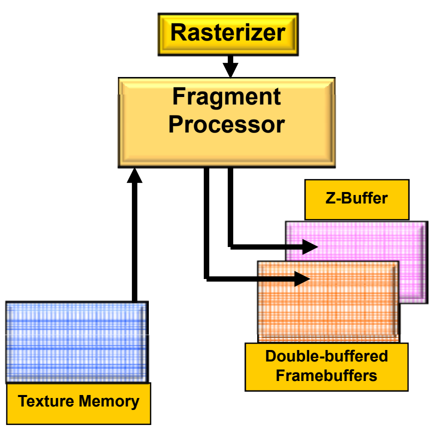
Display Resolution

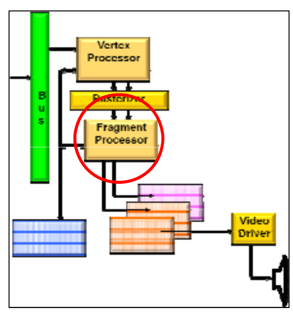
- ***Pixel*** resolutions (1024x768 - 1920x1152 are common)
- Screen size (13", 16", 19", 21" are common)
- Human acuity: 1 arc-minute is achieved by viewing a 19" monitor with 1280x1024 resolution from a distance of ~40 inches



http://en.wikipedia.org/wiki/File:Pixel_geometry_01_Pengo.jpg

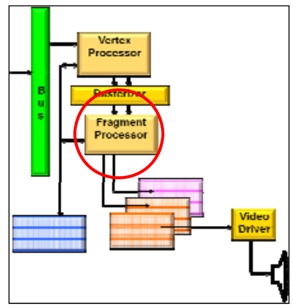
The Fragment Processor



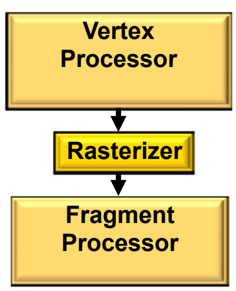


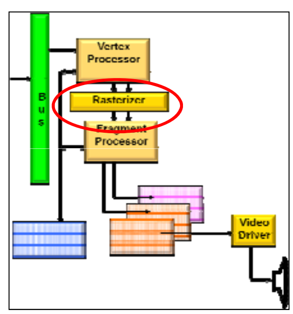
The Fragment Processor

- Takes in all information that describes this pixel
- Produces the RGBA for that pixel’s location in the framebuffer



The Rasterizer

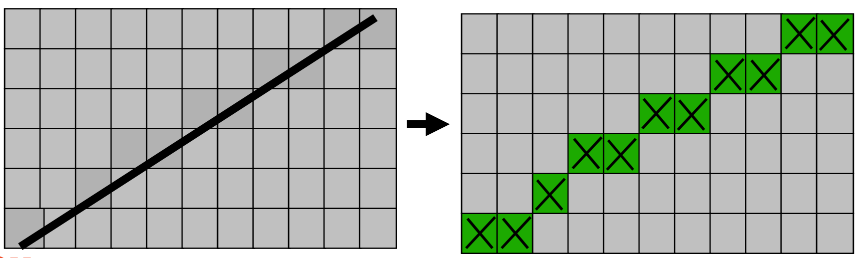




Oregon State University
Computer Graphics
Brown Cunningham
Associates

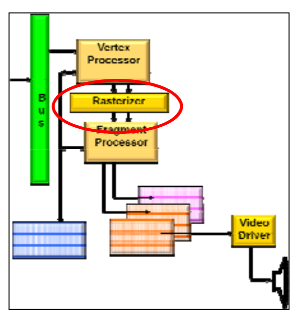
Rasterization

- Turn screen space vertex coordinates into pixels that make up lines and polygons
- A great place for custom electronics

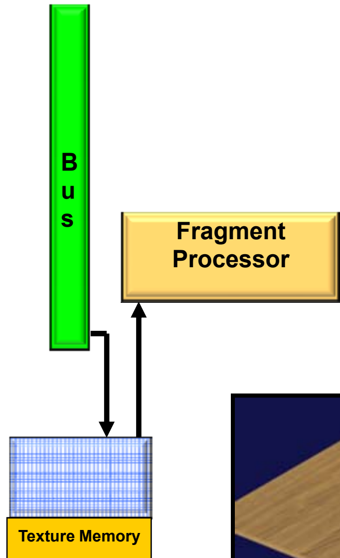


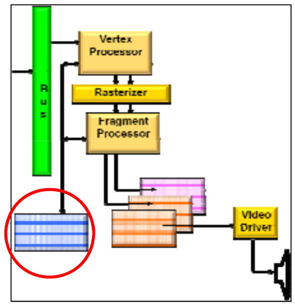
Rasterizers Can Interpolate:

- **X and Y**
- **Red-green-blue values**
- **Alpha values**
- **Z values**
- **Intensities**
- **Surface normals**
- **Texture coordinates**
- **Custom values given by the shaders**



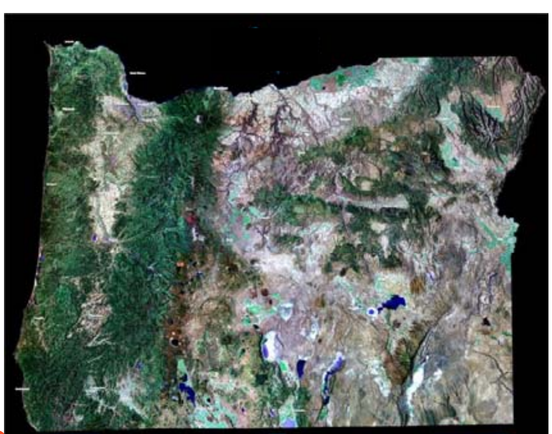
Texture Mapping

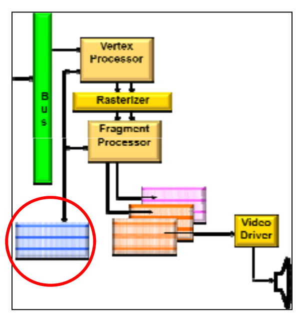




Texture Mapping

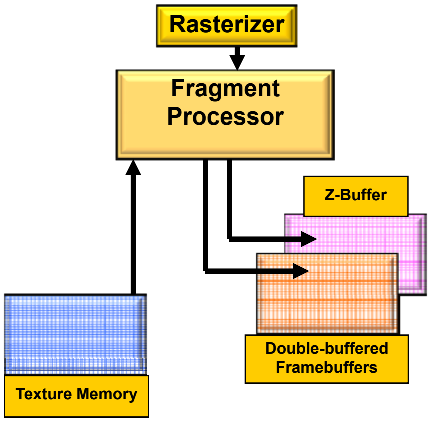
- “Stretch” an image onto a piece of geometry
- Image can be generated by a program or scanned in
- Useful for realistic scene generation

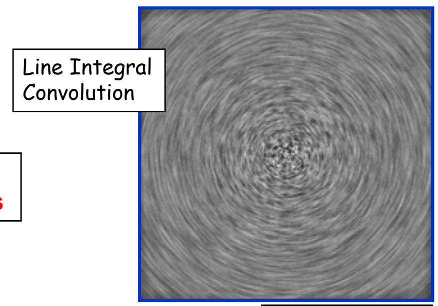




http://2ols.com

Something Cool: Write-Your-Own Fragment-Processor Code

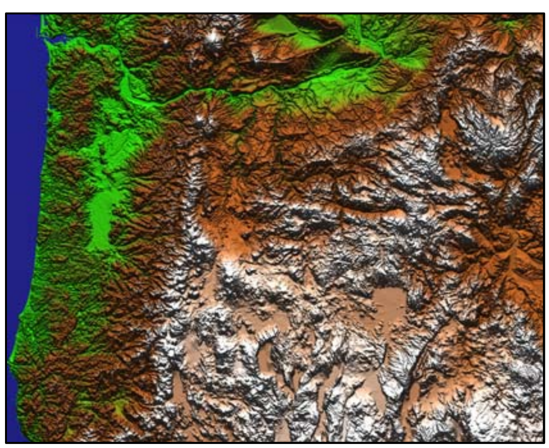




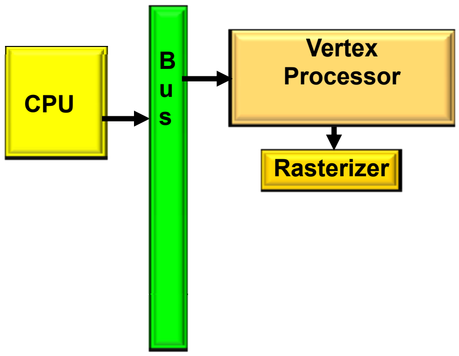
Referred to as:
Pixel Shaders or Fragment Shaders

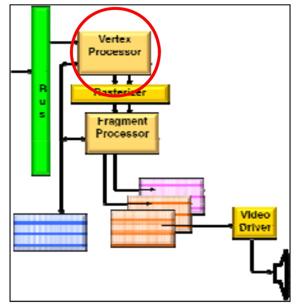
Procedural Texture Mapping

Create a texture from an equation. In this case, the equation takes a grid of heights and produces surface normals for lighting



The Vertex Processor



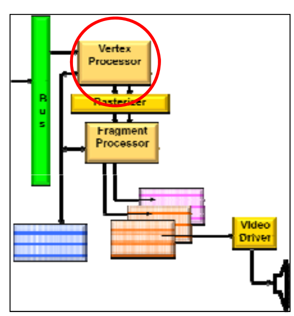


Oregon State University
Computer Graphics

Brown Cunningham
Associates

Vertex Processor

- Coordinates enter in model units
- Coordinates leave in screen (pixel) units
- Another great place for custom electronics

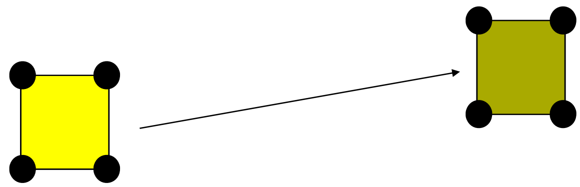


Oregon State University
Computer Graphics

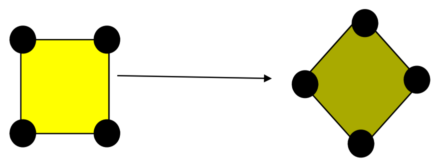
Brown Cunningham
Associates

Vertex Processor: Transformations

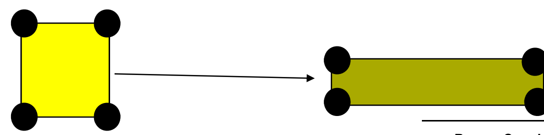
- Used to correctly place objects in the scene
- Translation



- Rotation



- Scaling

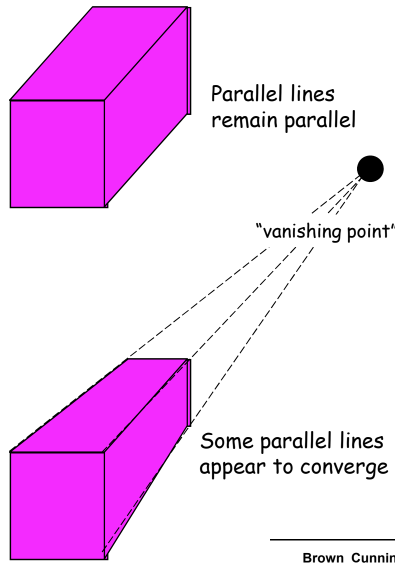


Vertex Processor: Windowing and Clipping

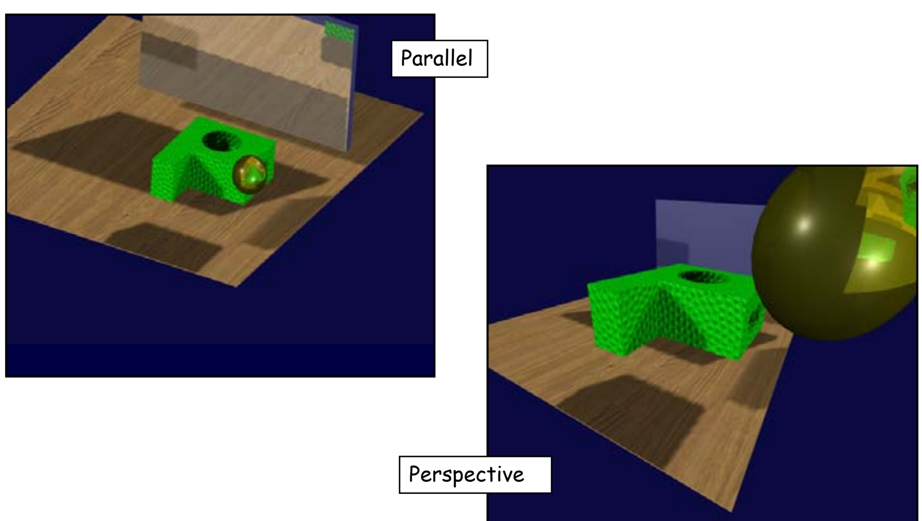
- Declare which portion of the 3D universe you are interested in viewing
- This is called the ***view volume***
- Clip away everything that is outside the viewing volume

Vertex Processor: Projection

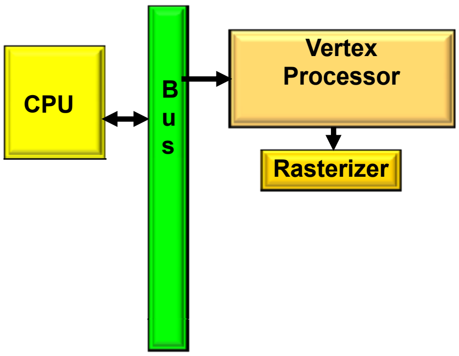
- **Turn 3D coordinates into 2D**
  - ***Parallel projection***
  - ***Perspective projection***

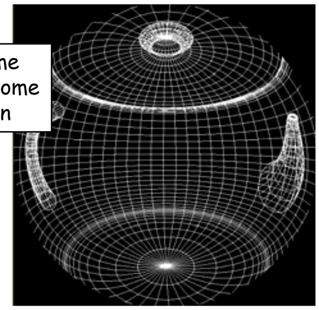


Vertex Processor: Projection

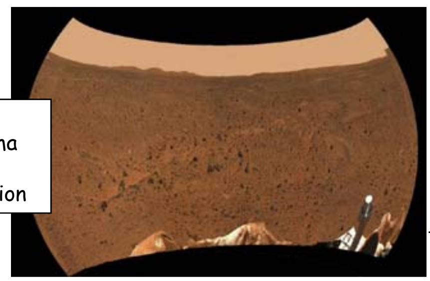


Something Cool:
Write-Your-Own Vertex Code





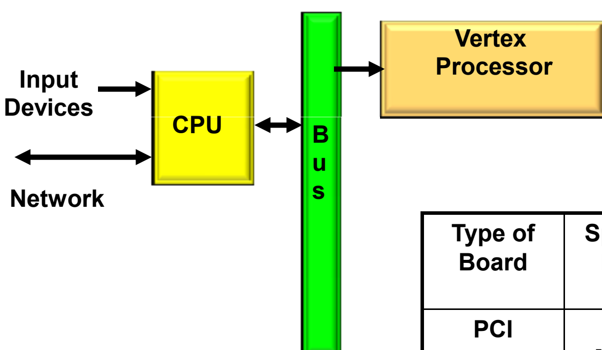
Wireframe Teapot Dome Projection



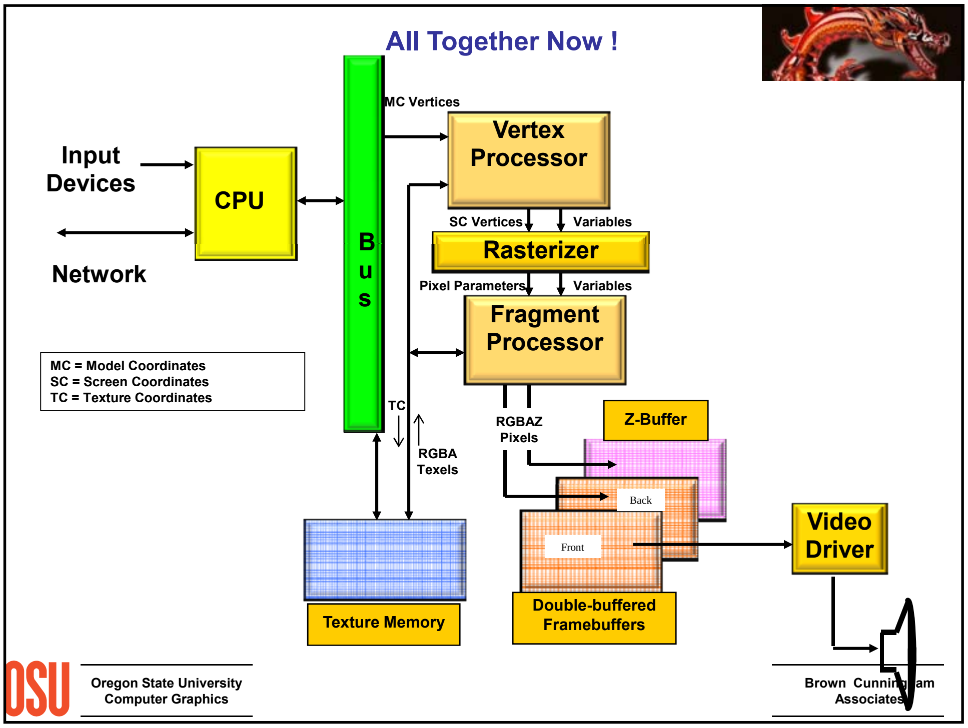
Mars Panorama Dome Projection

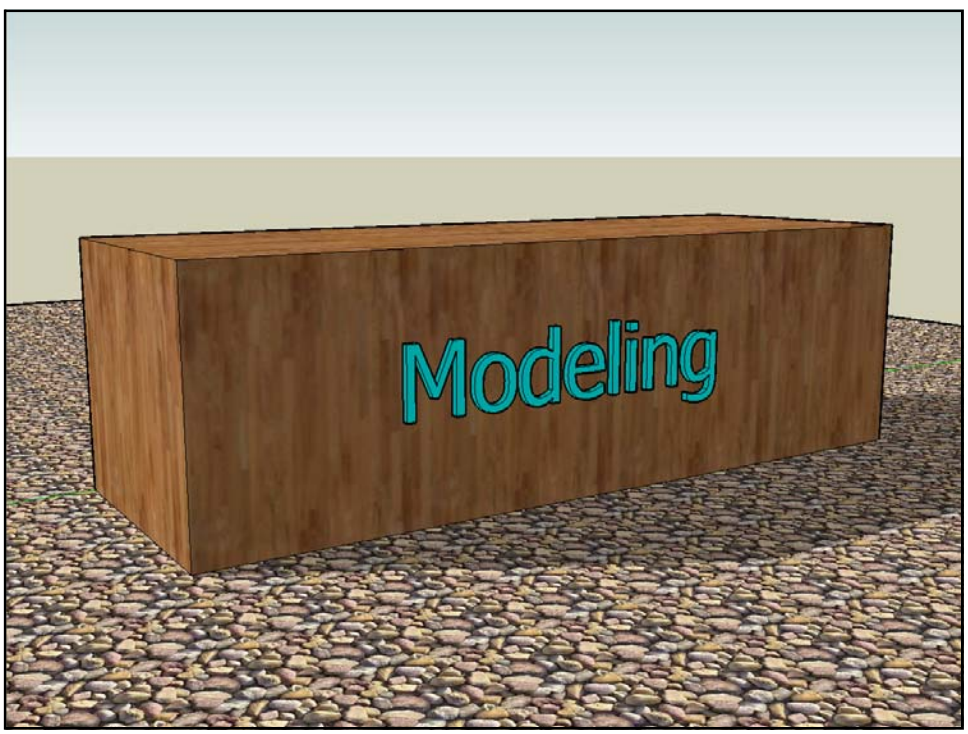
Referred to as:
Vertex Shaders

The CPU and Bus



Type of Board	Speed to Board	Speed from Board
PCI	132 Mb/sec	132 Mb/sec
AGP 8X	2 Gb/sec	264 Mb/sec
PCI Express	4 Gb/sec	4 Gb/sec


All Together Now !
MC Vertices
Vertex Processor
Input Devices
CPU
Network
B u s
SC Vertices
Variables
Rasterizer
Pixel Parameters
Variables
Fragment Processor
MC = Model Coordinates
SC = Screen Coordinates
TC = Texture Coordinates
TC
RGBA Texels
RGBAZ Pixels
Z-Buffer
Back
Front
Video Driver
Texture Memory
Double-buffered Framebuffers
OSU
Oregon State University
Computer Graphics


Modeling

What is a Model?

A is a model of B if A can be used to ask questions about B.

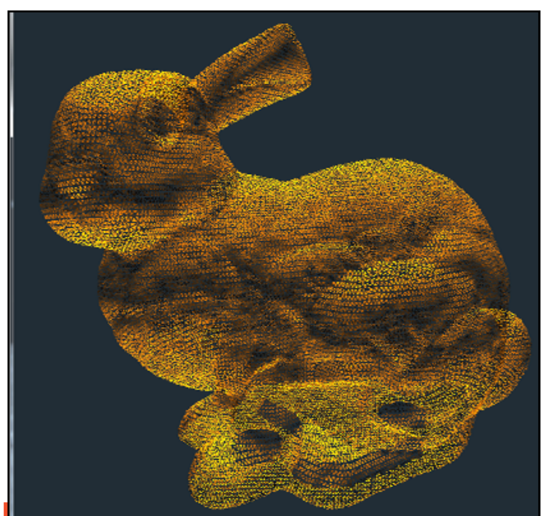
In computer graphics applications, what do we want to ask about B?

- What does B look like?
- How do I want to interact with (shape) B?
- Does B need to be a legal solid?
- How does B interact with its environment?
- What is B's surface area and volume?

These questions, and answers, control what type of geometric modeling you need to do

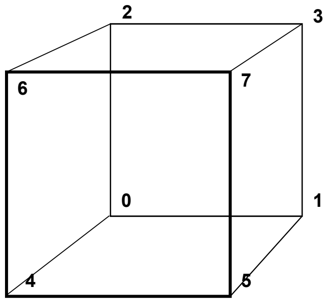
Explicitly Listing Geometry and Topology

Models can consist of thousands of vertices and faces – we need some way to list them efficiently



http://graphics.stanford.edu/data/3Dscanrep

Explicitly Listing Geometry and Topology

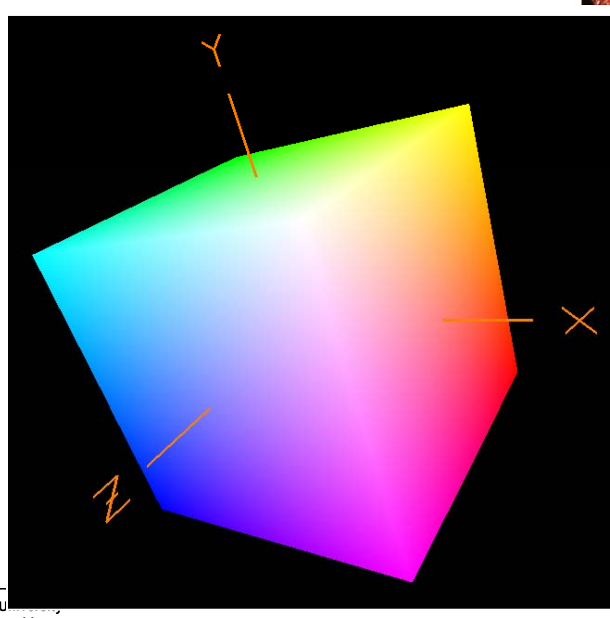



```
static GLfloat CubeVertices[ ][3] =
{
        { -1., -1., -1. },
        {  1., -1., -1. },
        { -1.,  1., -1. },
        {  1.,  1., -1. },
        { -1., -1.,  1. },
        {  1., -1.,  1. },
        { -1.,  1.,  1. },
        {  1.,  1.,  1. }
};
```

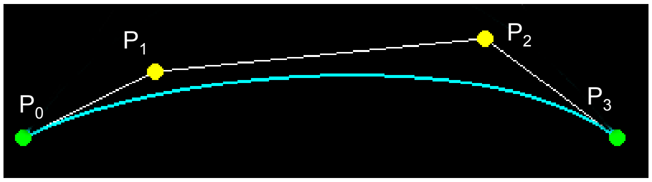
```
static GLfloat CubeColors[ ][3] =
{
        { 0., 0., 0. },
        { 1., 0., 0. },
        { 0., 1., 0. },
        { 1., 1., 0. },
        { 0., 0., 1. },
        { 1., 0., 1. },
        { 0., 1., 1. },
        { 1., 1., 1. },
};
```

```
static GLuint CubeIndices[ ][4] =
{
        { 0, 2, 3, 1 },
        { 4, 5, 7, 6 },
        { 1, 3, 7, 5 },
        { 0, 4, 6, 2 },
        { 2, 6, 7, 3 },
        { 0, 1, 5, 4 }
};
```


Cube Example



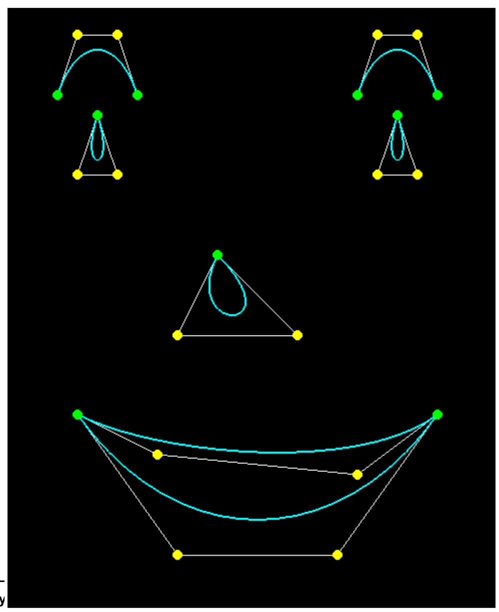
Curve Sculpting – Bezier Curve Sculpting



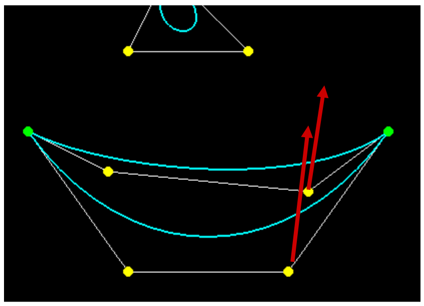
$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

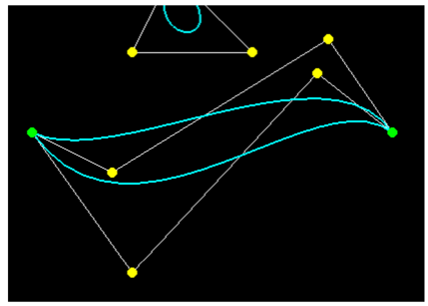
$$0. \le t \le 1.$$

Curve Sculpting – Bezier Curve Sculpting Example

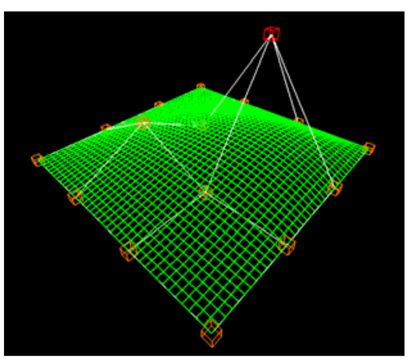


Curve Sculpting – Bezier Curve Sculpting Example

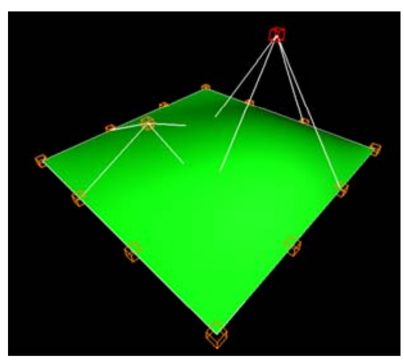




Surface Sculpting

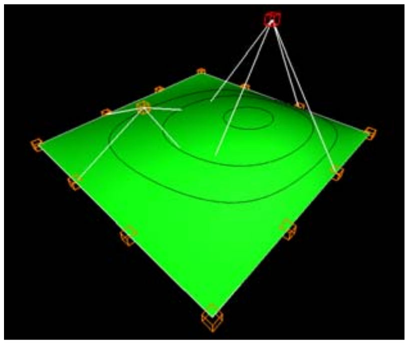


Wireframe

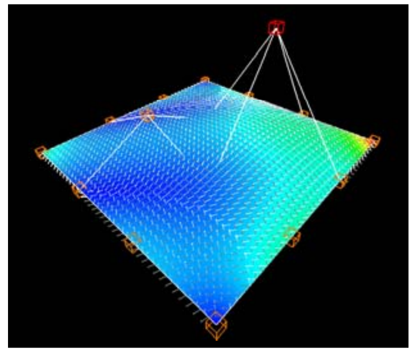


Surface

Surface equations can also be used for Analysis

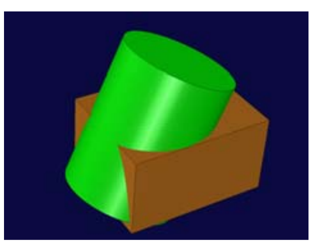


With Contour Lines

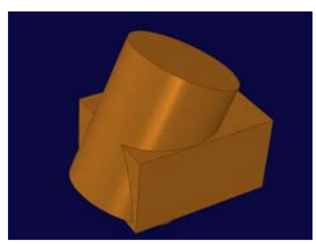


Showing Curvature

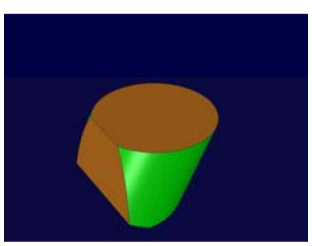
Solid Modeling Using Boolean Operators



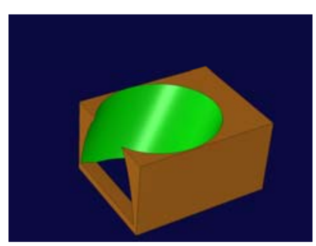
Two Overlapping Solids



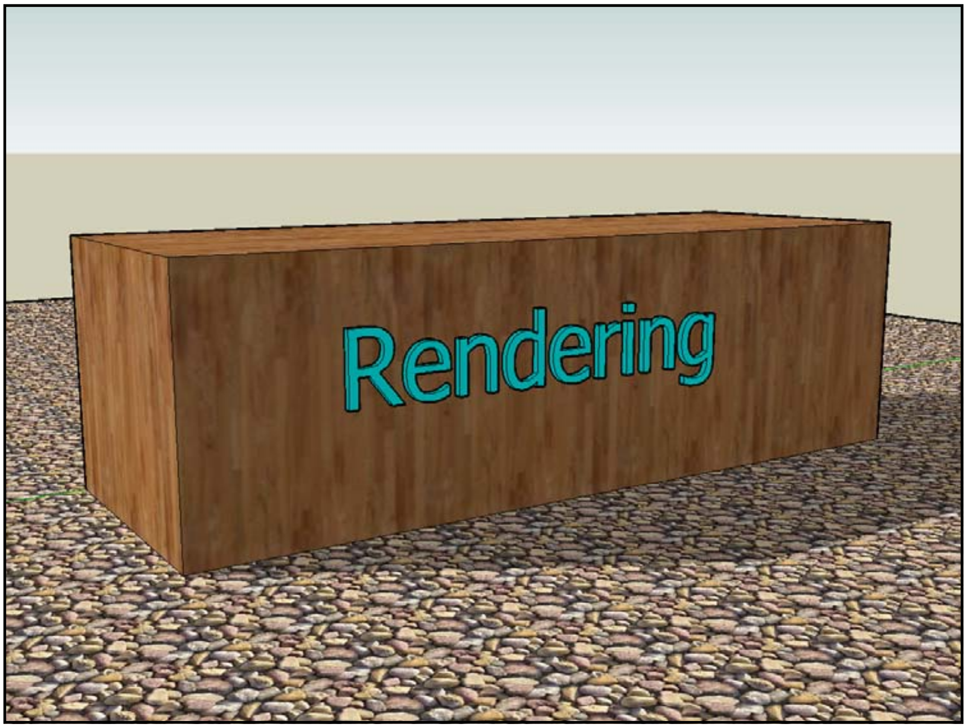
Union



Intersection



Difference

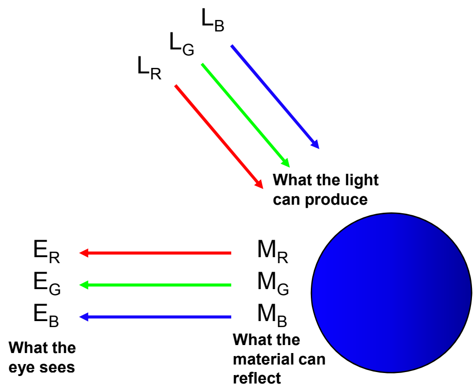


Rendering

Rendering is the process of creating an image of a geometric model. Again, there are questions you need to ask:

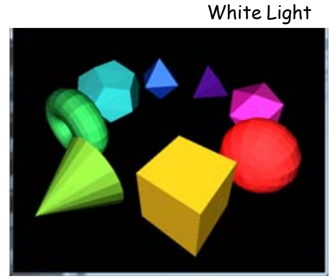
- How realistic do I want this image to be?
- How much compute time do I have to create this image?
- Do I need to take into account lighting?
- Does the illumination need to be global or will local do?
- Do I need to take into account shadows?
- Do I need to take into account reflection and refraction?

Fundamentals of Computer Graphics Lighting

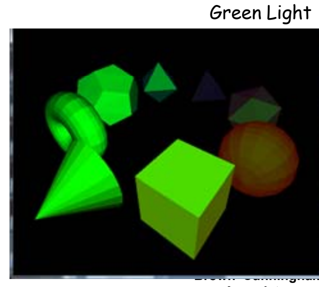


$$E_R = L_R * M_R$$
$$E_G = L_G * M_G$$
$$E_B = L_B * M_B$$

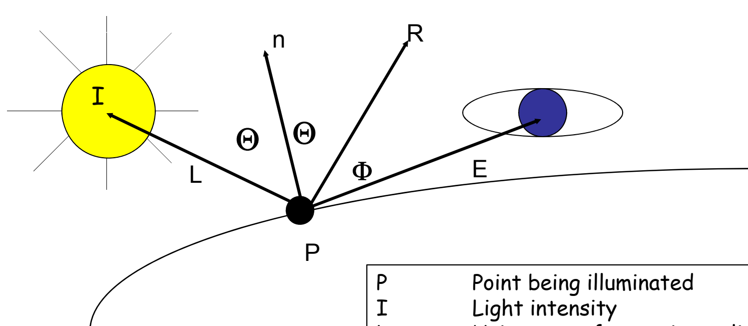
White Light



Green Light



The Computer Graphics Lighting Environment



P	Point being illuminated
I	Light intensity
L	Unit vector from point to light
n	Unit vector surface normal
R	Perfect reflection unit vector
E	Unit vector to eye position

Three Elements of Computer Graphics Lighting

1. Ambient = a constant — Accounts for light bouncing "everywhere"

2. Diffuse = $I * \cos\Theta$ — Accounts for the angle between the incoming light and the surface normal

3. Specular = $I * \cos^{S}\phi$ — Accounts for the angle between the "perfect reflector" and the eye; also the exponent, S, accounts for surface shininess

Note that $\cos\Theta$ is just the dot product between unit vectors L and n

Note that $\cos\phi$ is just the dot product between unit vectors R and E

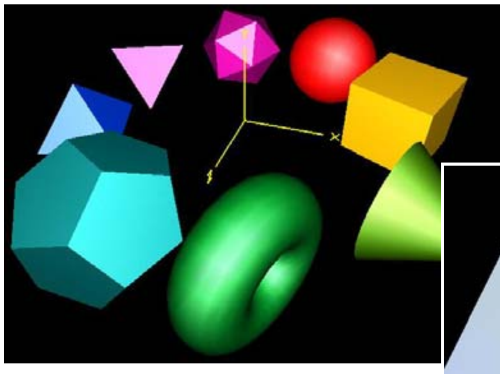
+

+

=

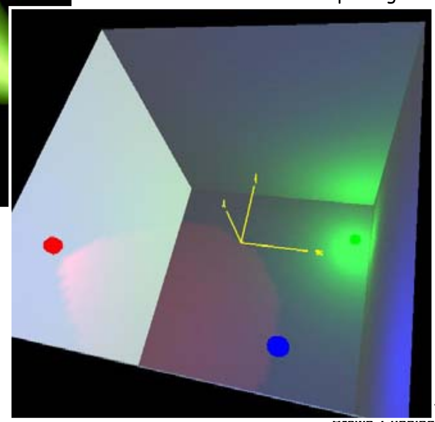
Three Elements of Computer Graphics Lighting

Lighting Examples



Omnidirectional Point Light

Spot Lights

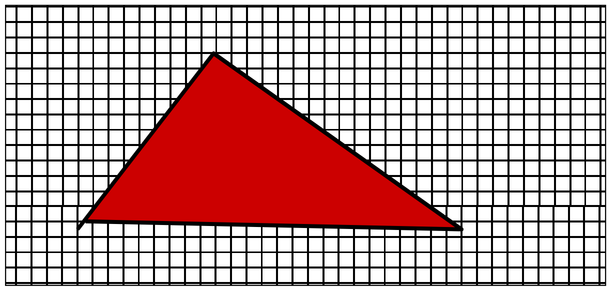


Two Types of Rendering

1. Starts at the object
2. Starts at the eye

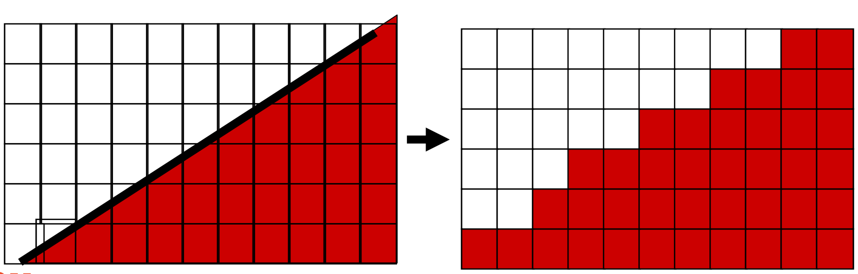
Starts at the Object

This is the typical kind of rendering you get on a graphics card.
Start with the geometry and project it onto the pixels.



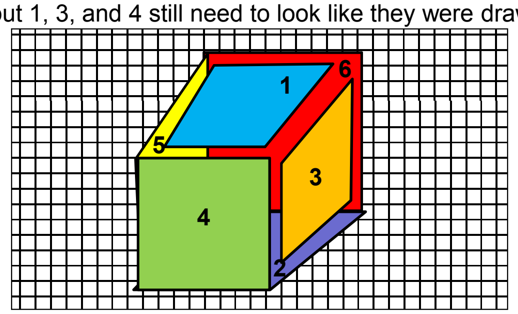
Rasterization

- **Turn screen space vertex coordinates into pixels that make up lines and polygons**
- **A great place for custom electronics**

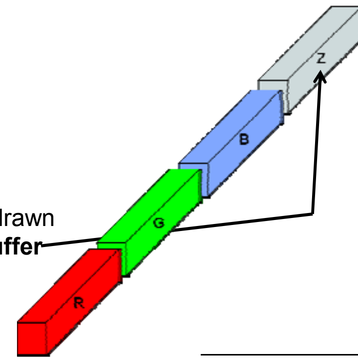


How do things in front *look* like they are really in front?

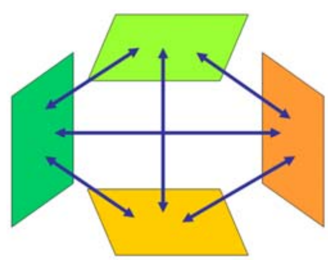
Your application might draw the polygons in 1-2-3-4-5-6 order, but 1, 3, and 4 still need to look like they were drawn last:



Either the polygons need to be re-arranged to be drawn in a back-to-front order, or we need to have a **Z-buffer**



Another From-the-Object Method -- Radiosity



Based on the idea that all surfaces gather light intensity from all other surfaces

The fundamental radiosity equation is an energy balance that says:

"The light energy leaving surface *i* equals the amount of light energy generated by surface *i* plus surface *i*'s reflectivity times the amount of light energy arriving from all other surfaces"

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$$

The Radiosity Equation

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$$

B_i is the light energy intensity shining from surface element *i*

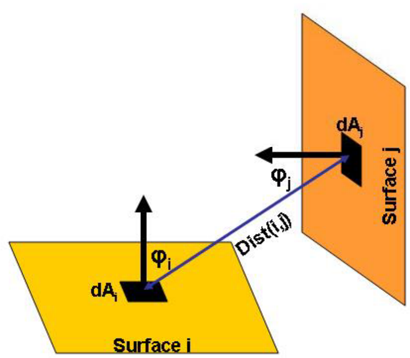
A_i is the area of surface element *i*

E_i is the internally-generated light energy intensity for surface element *i*

ρ_i is surface element *i*'s reflectivity

$F_{j \to i}$ is referred to as the Form Factor, or Shape Factor, and describes what percent of the energy leaving surface element *j* that arrives at surface element *i*

The Radiosity Shape Factor



$$F_{j \to i} = \int_{Ai} \int_{A_j} visibility(di, dj) \frac{\cos \Theta_i \cos \Theta_j}{\pi Dist(di, dj)^2} dA_j dA_i$$

The Radiosity Matrix Equation

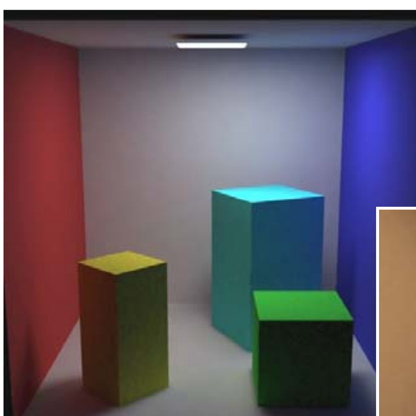
Expand $B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \rightarrow i}$

For each surface element, and re-arrange to solve for the surface intensities, the *B*'s:

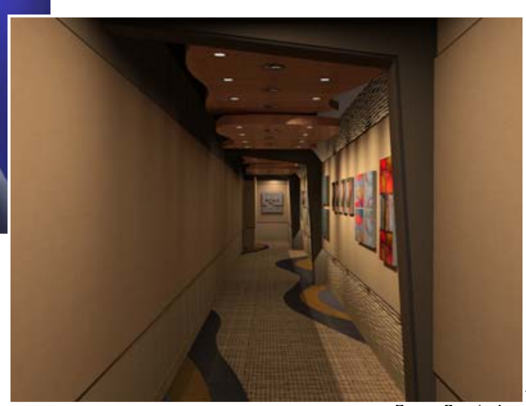
$$\begin{bmatrix} 1-\rho_1 F_{1\rightarrow 1} & -\rho_1 F_{1\rightarrow 2} & \bullet\bullet\bullet & -\rho_1 F_{1\rightarrow N} \\ -\rho_2 F_{2\rightarrow 1} & 1-\rho_2 F_{2\rightarrow 2} & \bullet\bullet\bullet & -\rho_2 F_{2\rightarrow N} \\ \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet \\ -\rho_N F_{N\rightarrow 1} & -\rho_N F_{N\rightarrow 2} & \bullet\bullet\bullet & 1-\rho_N F_{N\rightarrow N} \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ \bullet\bullet\bullet \\ B_N \end{Bmatrix} = \begin{Bmatrix} E_1 \\ E_2 \\ \bullet\bullet\bullet \\ E_N \end{Bmatrix}$$

This is a lot of equations!

Radiosity Examples



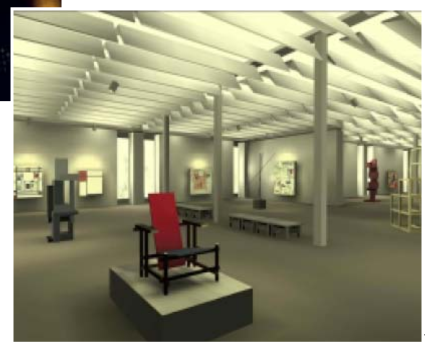
AR Toolkit



Autodesk

Radiosity Examples

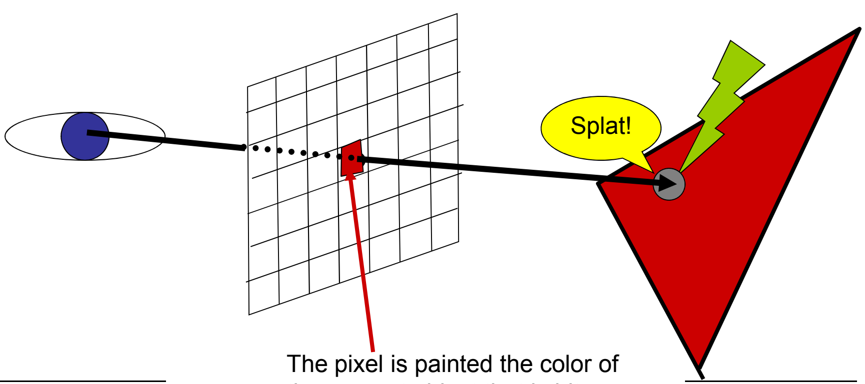
Cornell University



Cornell University

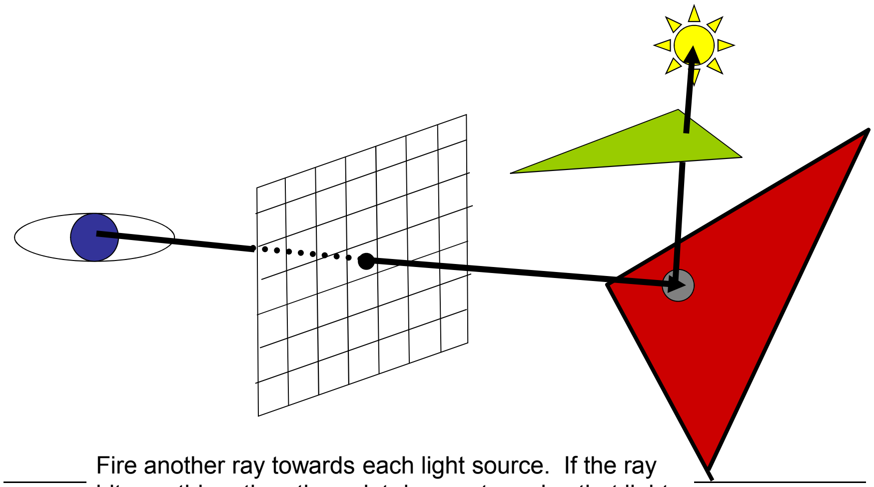
Starts at the Eye

The most common approach in this category is ray-tracing:



Starts at the Eye

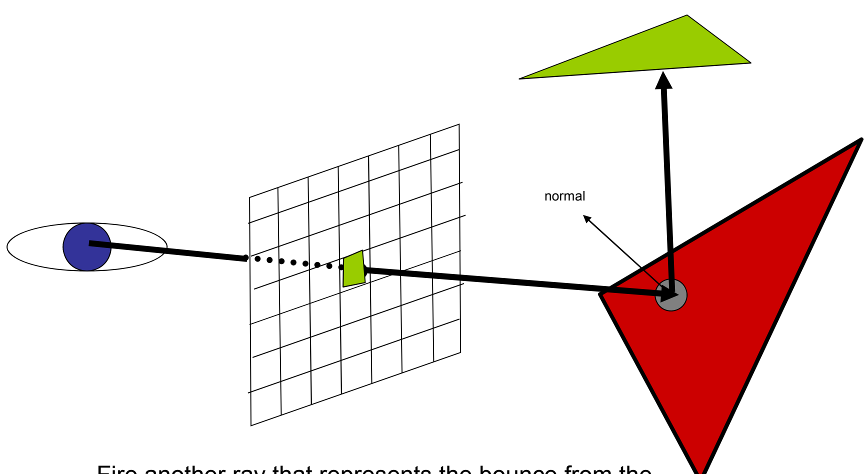
It's also easy to see if this point lies in a shadow:



Fire another ray towards each light source. If the ray hits anything, then the point does not receive that light.

Starts at the Eye

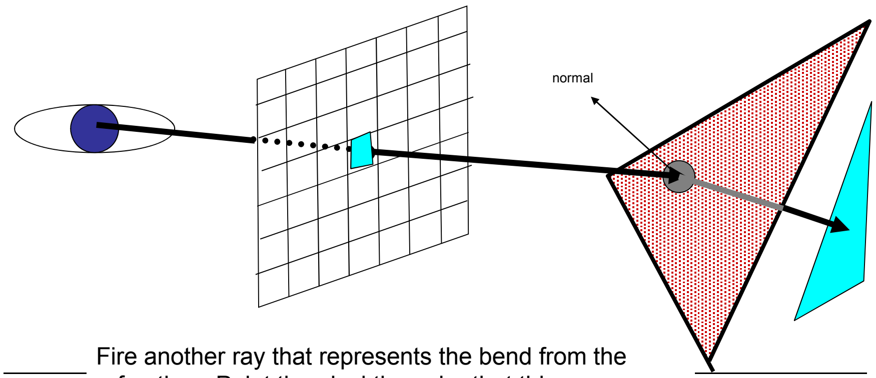
It's also easy to handle reflection



Fire another ray that represents the bounce from the reflection. Paint the pixel the color that this ray sees.

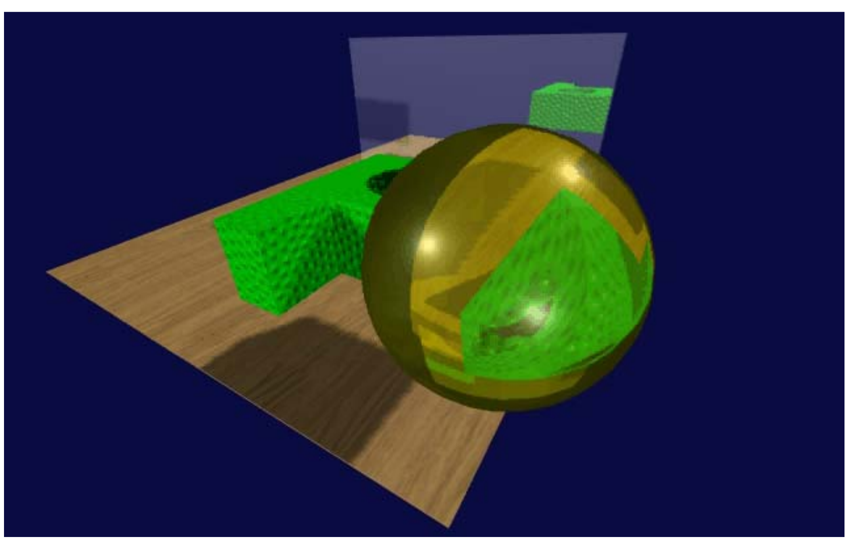
Starts at the Eye

It's also easy to handle refraction

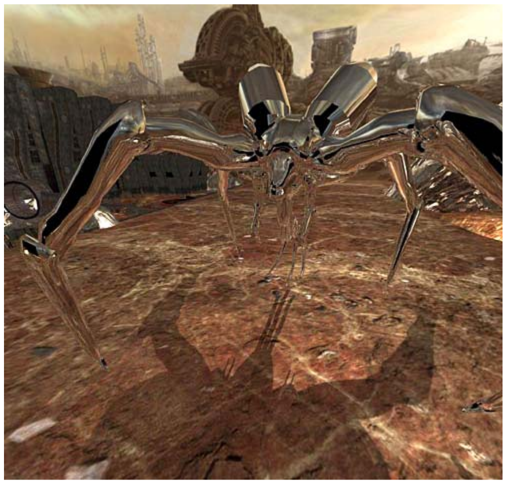


Fire another ray that represents the bend from the refraction. Paint the pixel the color that this ray sees.

Ray Tracing Examples



Ray Tracing Examples



Quake 4 Ray-Tracing Project

Oregon State University
Computer Graphics

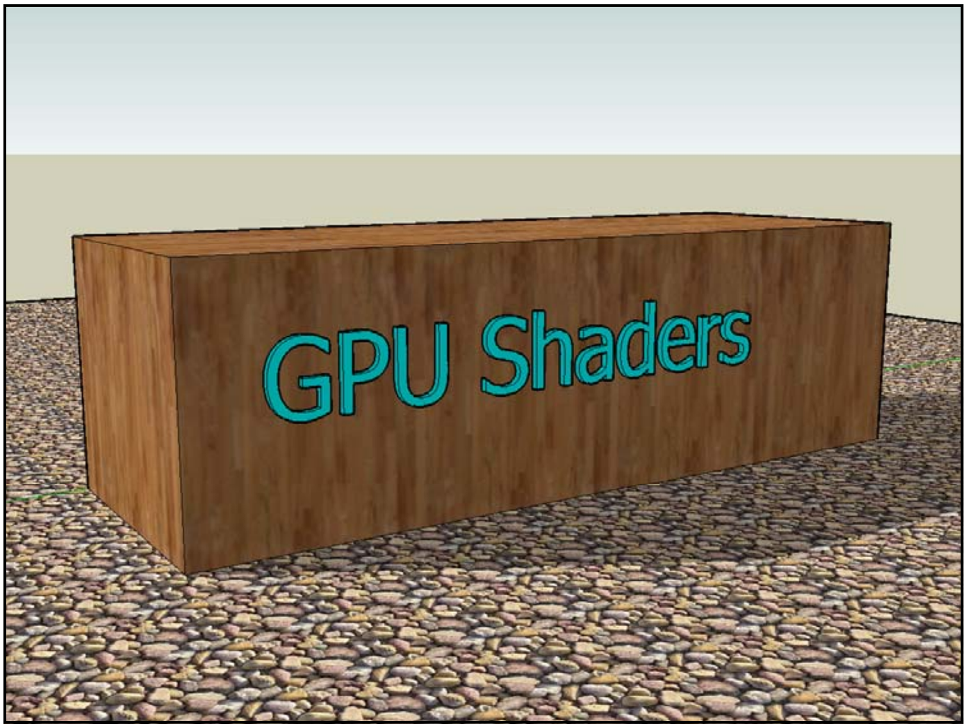
Brown Cunningham
Associates

Ray Tracing Examples

IBM's Cell Interactive Ray-tracer

Oregon State University
Computer Graphics

Brown Cunningham
Associates



GPU Shader Programming

• Allows programmers to load their own code into parts of the hardware graphics pipeline

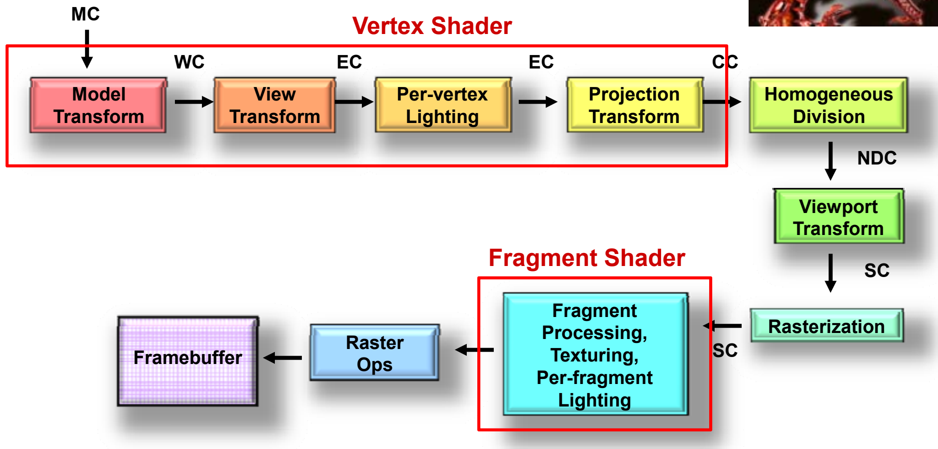
• Gives a unique combination of control and speed

• This is a hot, new area in computer graphics

• These notes will focus on *what* can be done this way, not on *how* to do it (that would take lots more time)

• If you want to know more, there’s another course on just this topic!

The Basic Computer Graphics Pipeline



MC = Model Coordinates
WC = World Coordinates
EC = Eye Coordinates
CC = Clip Coordinates
NDC = Normalized Device Coordinates
SC = Screen Coordinates

A GLSL Vertex Shader Replaces These Operations:

- Vertex transformations
- Normal transformations
- Normal normalization
- Handling of per-vertex lighting
- Handling of per-vertex colors
- Handling of texture coordinates

A GLSL Fragment Shader Replaces These Operations:

- Color computation
- Texturing
- Handling of per-pixel lighting
- Fog
- Blending
- Discarding fragments

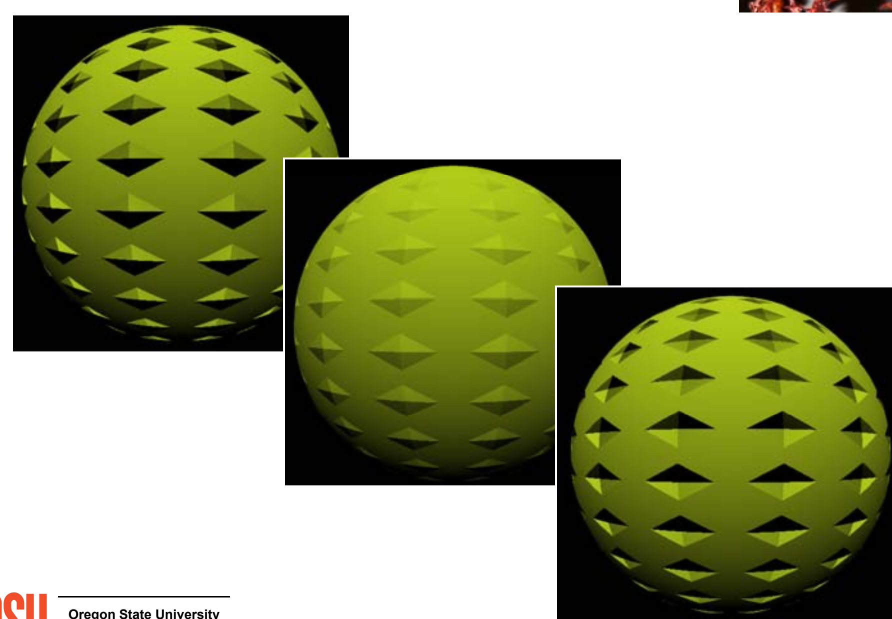
A GLSL Tessellation Shader:

- Breaks geometry into smaller pieces based on adjacent points, size, curvature, etc.

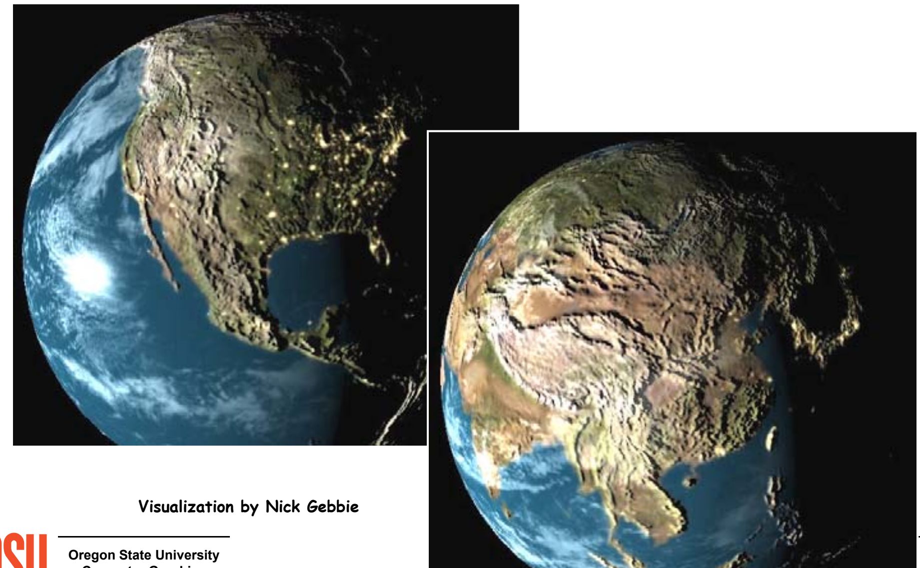
A GLSL Geometry Shader:

- Breaks geometry into smaller pieces based on more limited information
- Changes the geometry's topology type
- Changes the object's coordinates

Bump Mapping with Shaders



Bump Mapping with Shaders



Visualization by Nick Gebbie

Cube Mapping with Shaders

Cube Map of NVIDIA's Lobby

Cube Mapping with Shaders

Oregon State University
Computer Graphics

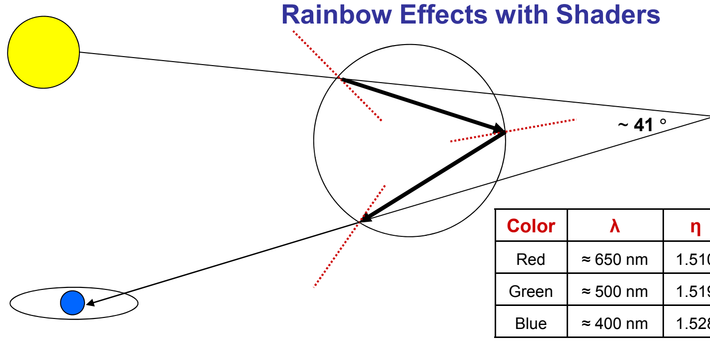
Brown Cunningham
Associates

Cube Mapping with Shaders

Oregon State University
Computer Graphics

Brown Cunningham
Associates

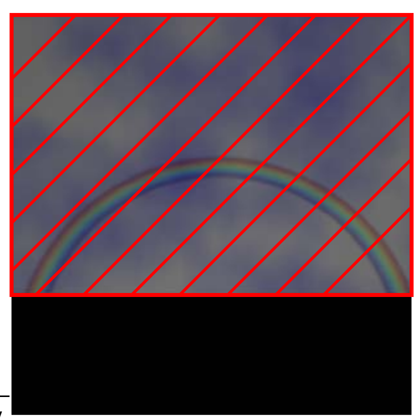
Rainbow Effects with Shaders



Color	λ	η	Θ	cosΘ	ΘΘ
Red	≈ 650 nm	1.510	42°	0.743	50.0°
Green	≈ 500 nm	1.519	41°	0.755	51.5°
Blue	≈ 400 nm	1.528	40°	0.766	53.0°

Rainbow Strategy

1. Draw one big quadrilateral across the scene
2. Anywhere that .7400 ≤ cos(Θ) ≤ .7700, paint the correct color
3. If not, discard that fragment




Finding Additional
Information

Where to Find More Information about Computer Graphics and Related Topics

Mike Bailey
Oregon State University

1. References

1.1 General Computer Graphics

SIGGRAPH Online Bibliography Database:
`http://www.siggraph.org/publications/bibliography`

Edward Angel and Dave Shreiner, *Interactive Computer Graphics: A Top-down Approach with OpenGL*, 6th Edition, Addison-Wesley, 2011.

Francis Hill and Stephen Kelley, *Computer Graphics Using OpenGL*, 3rd Edition, Prentice Hall, 2006.

Steve Cunningham, *Computer Graphics: Programming in OpenGL for Visual Communication*, Prentice-Hall, 2007

Alan Watt, *3D Computer Graphics*, 3rd Edition, Addison-Wesley, 2000.

Peter Shirley, *Fundamentals* of Computer Graphics, 2nd Edition, AK Peters, 2005.

Andrew Glassner, *Graphics Gems*, Academic Press, 1990.

James Arvo, *Graphics Gems 2*, Academic Press, 1991.

David Kirk, *Graphics Gems 3*, Academic Press, 1992.

Paul Heckbert, *Graphics Gems 4*, Academic Press, 1994.

Alan Paeth, *Graphics Gems 5*, Academic Press, 1995.

Jim Blinn, *A Trip Down the Graphics Pipeline*, Morgan Kaufmann, 1996.

Jim Blinn, *Dirty Pixels*, Morgan Kaufmann, 1998.

David Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, 1997.

SIGGRAPH Conference Final program.

1.2 Math and Geometry

Michael Mortenseon, *Geometric Transformations for 3D Modeling*, 2nd Edition, Industrial press, 2007.

Michael Mortenson, *Geometric Modeling*, John Wiley & Sons, 2006.

Eric Lengyel, *Mathematics for 3D Game Programming and Computer Graphics*, Charles River Media, 2002.

Jean Gallier, *Curves and Surfaces in Geometric Modeling*, Morgan Kaufmann, 2000.

Walter Taylor, *The Geometry of Computer Graphics*, Wadsworth & Brooks/Cole, 1992.

Gerald Farin, *Curves and Surfaces for Computer Aided Geometric Design*, 3rd Edition, Academic Press, 2001.

Gerald Farin and Dianne Hansford, *The Geometry Toolbox for Graphics and Modeling*, AK Peters, 1998.

Joe Warren and Henrik Weimer, *Subdivision Methods for Geometric Design: A Constructive Approach*, Morgan Kaufmann, 2001.

Barrett O'Neil, *Elementary Differential Geometry*, Academic Press, 1997.

Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1996.

Christopher Hoffman, *Geometric & Solid Modeling*, Morgan Kaufmann, 1989.

I.D. Faux and M.J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis-Horwood, 1979.

Eric Stollnitz, Tony DeRose, and David Salesin, *Wavelets for Computer Graphics*, Morgan-Kaufmann, 1996.

Ronen Barzel, *Physically-Based Modeling for Computer Graphics*, Academic Press, 1992.

David Rogers and J. Alan Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill, 1989.

John Snyder, *Generative Modeling for Computer Graphics and Computer Aided Design*, Academic Press, 1992.

1.3 Scientific Visualization

Klaus Engel, Markus Hadwiger, Joe Kniss, Christof Rezk-Salama, and Daniel Weiskopf, *Real-Time Volume Graphics*, A.K. Peters, 2006.

Christopher Johnson and Charles Hansen, *The Visualization Handbook*, Elsevier Academic Press, 2005.

David Thompson, Jeff Braun, and Ray Ford, *OpenDX: Paths to Visualization*, Visualization and Imagery Solutions, Inc., 2001.

Chandrajit Bajaj, *Data Visualization Techniques*, John Wiley & Sons, 1999.

Min Chen, Arie Kaufman, and Roni Yagel, *Volume Graphics*, Springer-Verlag, 2000.

William Schroeder, Ken Martin, and Bill Lorensen, *The Visualization Toolkit*, 3rd Edition, Prentice-Hall, 2004.

Luis Ibanez and William Schroeder, *The ITK Software Guide: The Insight Segmentation and Registration Toolkit (version 1.4)*, Prentice-Hall, 2003.

Greg Nielson, Hans Hagen, and Heinrich Müller, *Scientific Visualization: Overviews, Methodologies, Techniques,* IEEE Computer Society Press, 1997.

Lenny Lipton, *The CrystalEyes Handbook*, StereoGraphics Corporation, 1991.

Brand Fortner, *The Data Handbook: A Guide to Understanding the Organization and Visualization of Technical Data*, Spyglass, 1992.

William Kaufmann and Larry Smarr, *Supercomputing and the Transformation of Science,* Scientific American Library, 1993.

Robert Wolff and Larry Yaeger, *Visualization of Natural Phenomena*, Springer-Verlag, 1993.

Peter Keller and Mary Keller, *Visual Cues: Practical Data Visualization*, IEEE Press, 1993.

1.4 Shaders

Mike Bailey and Steve Cunningham, *Computer Graphics Shaders: Theory and Practice*, Second Edition, AK Peters, 2011.

Randi Rost, Bill Licea-Kane, Dan Ginsburg, John Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen, *OpenGL Shading Language,* Addison-Wesley, 2009. (3rd Edition)

Steve Upstill, *The RenderMan Companion*, Addison-Wesley, 1990.

Tony Apodaca and Larry Gritz, *Advanced RenderMan: Creating CGI for Motion Pictures*, Morgan Kaufmann, 1999.

Saty Raghavachary, *Rendering for Beginners: Image Synthesis using RenderMan*, Focal Press, 2005.

Randima Fernando, *GPU Gems*, NVIDIA, 2004.

Matt Pharr, Randima Fernando, *GPU Gems 2*, NVIDIA, 2005.

Hubert Nguyen, *GPU Gems 3*, NVIDIA, 2007.

`http://www.clockworkcoders.com/oglsl`

1.5 Gaming

`http://gamedeveloper.texterity.com/gamedeveloper/2008careerguide/`

David Hodgson, Bryan Stratten, and Alice Rush, *Paid to Play: An Insider's Guide to Video Game Careers*, Prima, 2006.

Alan Watt and Fabio Policarpo, *Advanced Game Development with Programmable Graphics Hardware*, AK Peters, 2005.

Jacob Habgood and Mark Overmars, *The Game Maker's Apprentice*, Apress, 2006.

David Eberly, *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*, Morgan Kaufmann, 2006.

Alan Watt and Fabio Policarpo, *3D Games: Real-time Rendering and Software Technology*, Addison-Wesley, 2001.

Eric Lengyel, *Mathematics for 3D Game Programming and Computer Graphics*, Charles River Media, 2002.

David Bourg, *Physics for Game Developers*, O'Reilly and Associates, 2002.

Munlo Coutinho, *Dynamic Simulations of Multibody Systems*, Springer Verlag, 2001.

Mark DeLoura, *Game Programming Gems,* Charles River Media, 2000.

Mark DeLoura, *Game Programming Gems 2*, Charles River Media, 2001.

Mark DeLoura, *Game Programming Gems 3*, Charles River Media, 2002.

`http://www.gamedev.net`

`http://www.gamasutra.net`

`http://www.yoyogame.com`

1.6 Color and Perception

Maureen Stone, *A Field Guide to Digital Color*, AK Peters, 2003.

Roy Hall, *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, 1989.

David Travis, *Effective Color Displays*, Academic Press, 1991.

L.G. Thorell and W.J. Smith, *Using Computer Color Effectively*, Prentice Hall, 1990.

Edward Tufte, *The Visual Display of Quantitative Information*, Graphics Press, 1983.

Edward Tufte, *Envisioning Information*, Graphics Press, 1990.

Edward Tufte, *Visual Explanations*, Graphics Press, 1997.

Edward Tufte, *Beautiful Evidence*, Graphics Press, 2006.

Howard Resnikoff, *The Illusion of Reality*, Springer-Verlag, 1989.

1.7 Rendering

Andrew Glassner, *Principles of Digital Image Synthesis*, Morgan Kaufmann, 1995.

Michael Cohen and John Wallace, *Radiosity and Realistic Image Synthesis*, Morgan-Kaufmann, 1993.

Andrew Glassner, *An Introduction to Ray Tracing*, Academic Press, 1989.

Rosalee Wolfe, *3D Graphics: A Visual Approach*, Oxford Press.

Ken Joy et al, *Image Synthesis*, IEEE Computer Society Press, 1988.

1.8 Images

David Ebert et al, *Texturing and Modeling*, 2nd Edition, Academic Press, 1998.

Alan Watt and Fabio Policarpo, *The Computer Image*, Addison-Wesley, 1998.

Ron Brinkman, *The Art and Science of Digital Compositing*, Morgan Kaufmann, 1999.

John Miano, *Compressed Image File Formats*, Addison-Wesley, 1999.

1.9 Animation

Alan Watt and Mark Watt, *Advanced Animation and Rendering Techniques*, Addison-Wesley, 1998.

Nadia Magnenat Thalmann and Daniel Thalmann, *Interactive Computer Animation*, Prentice-Hall, 1996.

Philip Hayward and Tana Wollen, *Future Visions: New Technologies of the Screen*, Indiana University Press, 1993.

1.10 Virtual Reality

John Vince, *Virtual Reality Systems*, Addison-Wesley, 1995.

1.11 The Web

Don Brutzman and Leonard Daly, *X3D: Extensible 3D Graphics for Web Authors*, Morgan Kaufmann, 2007

Rémi Arnaud and Mark Barnes, *Collada – Sailing the Gulf of 3D Digital Content Creation*, AK Peters, 2006.

Gene Davis, *Learning Java Bindings For OpenGL (JOGL)*, AuthorHouse, 2004.

Andrea Ames, David Nadeau, John Moreland, *The VRML 2.0 Sourcebook*, John Wiley & Sons, 1997.

Bruce Eckel, *Thinking in Java*, Prentice-Hall, 1998.

David Flanagan, *Java in a Nutshell*, O'Reilly & Associates, 5th edition, 2005.

David Flanagan, *Java Examples in a Nutshell*, O'Reilly & Associates, 3rd edition, 2004.

Henry Sowizral, Kevin Rushforth, and Michael Deering, *The Java 3D API Specification*, Addison-Wesley, 1998.

Rasmus Lerdorf and Kevin Tatroe, *Programming PHP*, O'Reilly, 2002.

Yukihiro Matsumoto, *Ruby in a Nutshell*, O'Reilly, 2003.

1.12 Stereographics

David McAllister, *Stereo Computer Graphics and Other True 3D Technologies*, Princeton University Press, 1993.

Shab Levy, *Stereoscopic Imaging: A Practical Guide*, Gravitram Creations, 2008.

1.13 Graphics Miscellaneous

OpenGL 3.0 Programming Guide, Addison-Wesley, 2009 (7th edition). (Eighth Edition coming?)

Aaftab Munshi, Dan Ginsburg, and Dave Shreiner, *OpenGL ES 2.0*, Addison-Wesley, 2008.

Tom McReynolds and David Blythe, *Advanced Graphics Programming Using OpenGL*, Morgan Kaufmann, 2005.

Edward Angel, *OpenGL: A Primer*, Addison-Wesley, 2009.

Andrew Glassner, *Recreational Computer Graphics*, Morgan Kaufmann, 1999.

Anne Spalter, *The Computer in the Visual Arts*, Addison-Wesley, 1999.

Jef Raskin, *The Humane Interface*, Addison-Wesley, 2000.

Ben Shneiderman, *Designing the User Interface*, Addison-Wesley, 1997.

Clark Dodsworth, *Digital Illusion*, Addison-Wesley, 1997.

Isaac Victor Kerlow, *The Art of 3-D: Computer Animation and Imaging*, 2000.

Isaac Victor Kerlow and Judson Rosebush, *Computer Graphics for Designers and Artists*, Van Nostrand Reinhold, 1986.

Mehmed Kantardzic, *Data Mining: Concepts, Models, Methods, and Algorithms*, Wiley, 2003.

William Press, Saul Teukolsky, William Vetterling, and Brian Flannery, *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1997.

James Skakoon and W. J. King, *The Unwritten Laws of Engineering*, ASME Press, 2001.

1.14 Software Engineering

Shari Lawrence Pfleeger and Joanne Atlee, *Software Engineering Theory and Practice*, Prentice Hall, 2006.

Tom Demarco and Timothy Lister, *Waltzing with Bears*, Dorset House Publishing, 2003.

Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

1.15 Parallel Programming

Peter Pacheco, *An Introduction to Parallel Programming*, Morgan-Kaufmann, 2011.

David B. Kirk, Wen-mei W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan-Kaufmann, 2010.

Maurice Herlihy and Nir Shavit, *The Art of Multiprocessor Programming*, Morgan Kaufmann, 2008.

James Reinders, *Intel Threading Building Blocks*, O'Reilly, 2007.

Bradford Nichols, Dick Buttlar, and Jacqueline Proudx Farrell, *Pthreads Programming*, O'Reilly, 1998.

Rohit Chandra, Leonardo Dagun, Dave Kohr, Dror Maydan, Jeff McDonald, Ramesh Menon, *Parallel Programming in OpenMP*, Morgan Kaufmann, 2001.

2. Periodicals

Computer Graphics and Applications: published by IEEE
(`http://www.computer.org`, 714-821-8380)

Computer Graphics World: published by Pennwell
(`http://www.cgw.com`, 603-891-0123)

Journal of Graphics, GPU, and Game Tools: published by A.K. Peters
(`http://www akpeters.com`, 617-235-2210)

Game Developer: published by CMP Media
(`http://www gdmag.com`, 415-905-2200)
(Once a year publishes the *Game Career Guide*.)

Computer Graphics Quarterly: published by ACM SIGGRAPH
(`http://www.siggraph.org`, 212-869-7440)

Computer Graphics Forum:, published by Eurographics
(http://www.eg.org/EG/Publications/CGF)

Computers & Graphics, published by Elsevier
(http://www.elsevier.com/locate/cag)

Transactions on Visualization and Computer Graphics: published by IEEE
(`http://www.computer.org`, 714-821-8380)

Transactions on Graphics: published by ACM
(`http://www.acm.org`, 212-869-7440)

Cinefex
(`http://www.cinefex.com`, 951-781-1917)

3. Professional organizations

ACM .... Association for Computing Machinery
`http://www.acm.org`
212-869-7440

SIGGRAPH ..... ACM Special Interest Group on Computer Graphics
`http://www.siggraph.org`
212-869-7440

EuroGraphics ... European Association for Computer Graphics
`http://www.eg.org`
Fax: +41-22-757-0318

IEEE................. Institute of Electrical and Electronic Engineers
`http://www.computer.org`
202-371-0101

IGDA .... International Game Developers Association
`http://www.igda.org`
856-423-2990

SIGCHI .... ACM Special Interest Group on Computer-Human Interfaces
`http://www.acm.org/sigchi`
212-869-7440

NAB................. National Association of Broadcasters
`http://www.nab.org`
800-521-8624

ASME .... American Society of Mechanical Engineers
`http://www.asme.org`
800-THE-ASME

4. Conferences

ACM SIGGRAPH:
2012: Los Angeles, CA – August 5-9
2013: Los Angeles, CA – July 28 – August 1
`http://www.siggraph.org/s2012`
`http://www.siggraph.org/s2013`

SIGGRAPH Asia:
2011: Hong Kong – December 12-15
`http://www.siggraph.org/asia2011`

IEEE Visualization:
2011: Providence, RI – October 23-28

http://visweek.org

Eurographics
2012: Cagliari, Italy – May 13-18
http://www.eurographics2012.it

Game Developers Conference:
2012: San Francisco, CA – March 5 - 9
http://www.gdconf.com

E3Expo
2012: Los Angeles, CA – June 7-9
http://www.e3expo.com

PAX (Penny Arcade Expo)
2011: Seattle, WA – August 26-28
http://www.paxsite.com

ASME International Design Engineering Technical Conferences (includes the Computers and Information in Engineering conference):
2012: Chicago, IL – August 12-15
http://www.asmeconferences.org/idetc2012

National Association of Broadcasters (NAB):
2012: Las Vegas, NV – April 14-19
http://www.nab.org

ACM SIGCHI:
2012: Austin, TX – May 5-10
http://www.acm.org/sigchi

ACM SIGARCH / IEEE Supercomputing:
2011: Seattle, WA -- November 12-18
http://www.supercomputing.org

5. Graphics Performance Characterization

The GPC web site tabulates graphics display speeds for a variety of vendors' workstation products. To get the information, visit:

http://www.spec.org/benchmarks.html#gwpg