



Camera Control in Computer Graphics: Models, Techniques and Applications

Course Notes

Siggraph Asia 2009

Marc Christie, INRIA Rennes Bretagne Atlantique, France
Patrick Olivier, Newcastle University, United Kingdom

Contents

1	Camera Control in Computer Graphics	2
	<i>Models, Techniques and Applications</i>	
1.1	Summary	2
1.2	Course Abstract	2
1.3	Intended Audience	3
1.4	Prerequisites	3
1.5	Presenters	3
1.6	Interest	4
2	Syllabus	5
2.1	Session 1: Introduction to camera control (slides 1 – 17)	5
2.2	Session 2: Photography and cinematography (slides 18 – 43)	5
2.3	Session 3: Interactive camera control (slides 44 – 73)	5
2.4	Session 4: Automated camera control (slides 74 – 104)	6
2.5	Session 5: Handling occlusion (slides 105 – 136)	6
2.6	Session 6: Trends in and open issues (slides 137 – 146)	6
3	Annotated Slides	7
4	Reference Papers	153
4.1	Paper 1: [HHS01]	153
4.2	Paper 2: [AVF04]	164
4.3	Paper 3: [KKS 05]	175
4.4	Paper 4: [GW92]	184
5	Acknowledgements	195

Chapter 1

Camera Control in Computer Graphics

Models, Techniques and Applications

Category

Computer Animation

1.1 Summary

Camera control is required in nearly all interactive 3D applications and presents a particular combination of technical challenges for which there have been a number of recent proposals. This course presents current and novel research ideas to the management of a user's viewpoint on a scene in interactive approaches, semi-automatic camera positioning, and fully declarative approaches, covering a range of techniques from specific path-planning, management of occlusion and modelling of high-level communicative goals.

1.2 Course Abstract

Our aim is to characterise the spectrum of applications that require automated camera control, present a summary of state-of-the-art models and techniques, and identify both promising avenues and hot topics for future research. Importantly, our presentation will be rigorous and synthetic, classifying the numerous techniques and identifying the representational limits and representational commitments of each.

Approaches range from completely interactive techniques that find their value in the possible mappings they propose between the user input and the camera coordinates, to complete automated paradigms whereby the camera moves and jumps according to high-level scenario-oriented goals. Between these extremes exists the automated approaches, with limited expressiveness, that utilise algebraic techniques and constraint-based optimisation. Our presentation of these approaches will include numerous live examples from both commercial systems and research prototypes.

A specific part of the course will be dedicated to current tough issues such as the proper handling of visibility for complex and/or multiple targets in dynamic environments.

The audience will get a broad overview of the state of the art in the domain and will have a range of detailed techniques that can be easily implemented.

1.3 Intended Audience

In addition to graphics researchers, who have a general interest in many contemporary issues in computer graphics, we anticipate interest from:

- computer games designers and developers;
- data visualisation and educational software developers and practitioners;
- graphics hardware vendors;
- people with an interest in the application of film theories to computer graphics.

1.4 Prerequisites

This course is accessible to any computer scientist involved in computer graphics. Though camera control is at the intersection of numerous domains (path-planning, visibility, view-point computation, cinematography), the course is built for beginners and offers a comprehensive overview of the domain. Most techniques are based on well-known fundamentals that will be recalled shortly (planning, viewpoint computation, optimization techniques) and the presentation will encompass numerous detailed illustrations and examples of applications.

1.5 Presenters

Marc Christie is an INRIA researcher in the Bunraku team at INRIA France and visiting professor from Nantes University. His research topic is located at the intersection of constraint solving techniques and computer graphics. He has been publishing contributions both in constraint and graphics communities related to the problem of declarative and real-time camera control systems. He has presented a State of the Art in Virtual Camera Control at Eurographics 2006 with Patrick Olivier and works on spatial and temporal partitioning techniques to characterize and reason on editing possibilities in virtual environments. He is a reviewer of well-known conferences (IJCAI, ECAI, EG, Siggraph).

Patrick Olivier is a senior lecturer in Newcastle University, UK. Dr Olivier has specific expertise in the link between artificial intelligence and computer graphics, in particular, automated reasoning about graphical representations. He is an active member of the artificial intelligence and graphics communities organising many workshops and symposia at AAAI, ECAI and IJCAI, he has edited several books on cognitive and computational aspects of spatial reasoning was the founding editor of the Journal of Spatial Cognition and Computation

and the co-editor with Steve Feiner of the Journal of Virtual Reality's recent special issue on Language, Speech and Gesture.

1.6 Interest

Camera control is an accessible, yet challenging, problem that any developer of interactive 3D graphics applications has encountered. Camera control frameworks must draw on both the mathematical rigor of spatial reasoning and path-planning, and the insights of cognitive science and psychological studies of visual perception and aesthetics.

Recently there has been a recent dramatic rise in interest in the field as proved by accepted publications in major events and journals: Wayfinder (EG2004), Semantic-space partitions (EG2005), Learning good views for intelligent galleries (EG2009), Viewpoint selection for intervention planning (IEEE VGTC 2007), Motion overview of human actions (Siggraph Asia 2008), Determination of camera parameters for character motions (The Visual Computer, 2008).

This rise in interest demonstrates the dynamism around the theme of camera control, coming from both an increasing need in presenting and navigating in large environments (data sets or realistic geometries), and increasing performances in CPU/GPU that foster possibilities that were before out of reach.

Chapter 2

Syllabus

2.1 Session 1: Introduction to camera control (slides 1 – 17)

We first introduce the motivations for automating camera control. We study three classes of application: computer games, modelling tools and multimodal systems. For each we identify key issues, the prevailing solutions (in outline) and their shortcoming. A key feature of our analysis of camera control in practice is the ad hoc nature of the formulations and solving techniques, and the complete absence of cinematic sophistication (i.e. no utilisation of anything resembling a grammar of film).

2.2 Session 2: Photography and cinematography (slides 18 – 43)

A direct insight can be offered by the use of real-world cameras from reports of photography and cinematography practice [Mascelli65, Katz91]. We describe in details the principles of camera positioning (where to set the camera) and camera composition (how to arrange elements on the screen). We introduce one of the key insights for camera control, that an algorithmic formulation of cinematic principles must always be performed in relation to an application domain (i.e. there is no general formulation).

2.3 Session 3: Interactive camera control (slides 44 – 73)

This session reviews the possible mappings of user inputs to camera parameters by referring to Ware and Osborne’s classification of interaction metaphors: camera in hand, world in hand and flying vehicle. Each metaphor has specialised applications, e.g. scene in hand allows to swivel around an object, the flying vehicle metaphor to navigate in large environments. A number of accounts we present have reported techniques to switch between these metaphors,

and have reported wider applicability. We further detail how interactive navigation approaches in complex environments such as virtual museums, buildings or factories strongly rely on path-planning results. The session will emphasize on representative applications and will detail the related algorithms at a real practical level.

2.4 Session 4: Automated camera control (slides 74 – 104)

This session covers the approaches that draw on the expressiveness of cinematography in the formulation of declarative camera control systems. The scene is described in terms of image properties which are in turn converted into relations between the degrees of freedom of the camera and the environment in order to construct adequate shots and camera paths. Such camera control problems can be formulated as either as a constraint-based system or as an optimisation problem, or more effectively as hybrid of both. Systems vary in their expressiveness (range and nature of image properties that are considered) and in the characteristics of the solving techniques (discrete or continuous nature, determinism, completeness).

2.5 Session 5: Handling occlusion (slides 105 – 136)

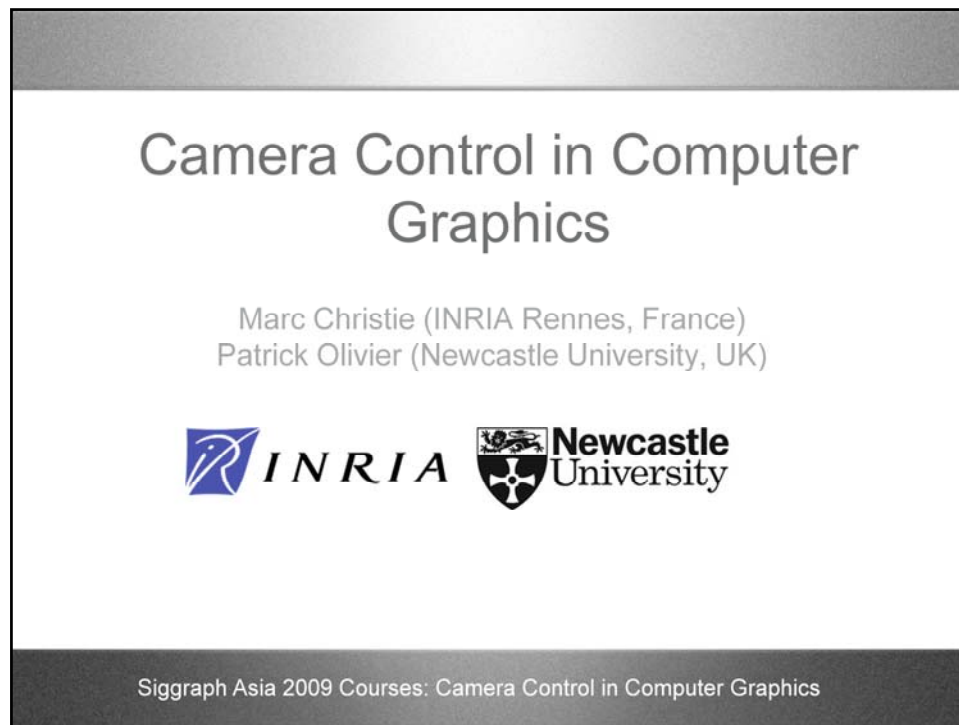
This session first explores the nature of occlusion (partial, temporal) over different representations of occluders (points, bounding volumes, exact geometry and representative shapes) and different visibility models. A detailed comparison is performed between ray-casting occlusion detection and hardware rendering techniques through numerous examples. We illustrate the strong correlation with penumbra maps and explore possible ways to adapt the models and techniques. We present recent results in the domain. The main computational issues are identified and we sketch promising research directions.

2.6 Session 6: Trends in and open issues (slides 137 – 146)

This final session discusses the problems related to the actual deployment of these techniques and directions for future research, including (1) augmenting the expressiveness by considering cognitively well-founded perceptual and aesthetic properties; and (2) the development of editing constraints (for discontinuous shot camera transitions).

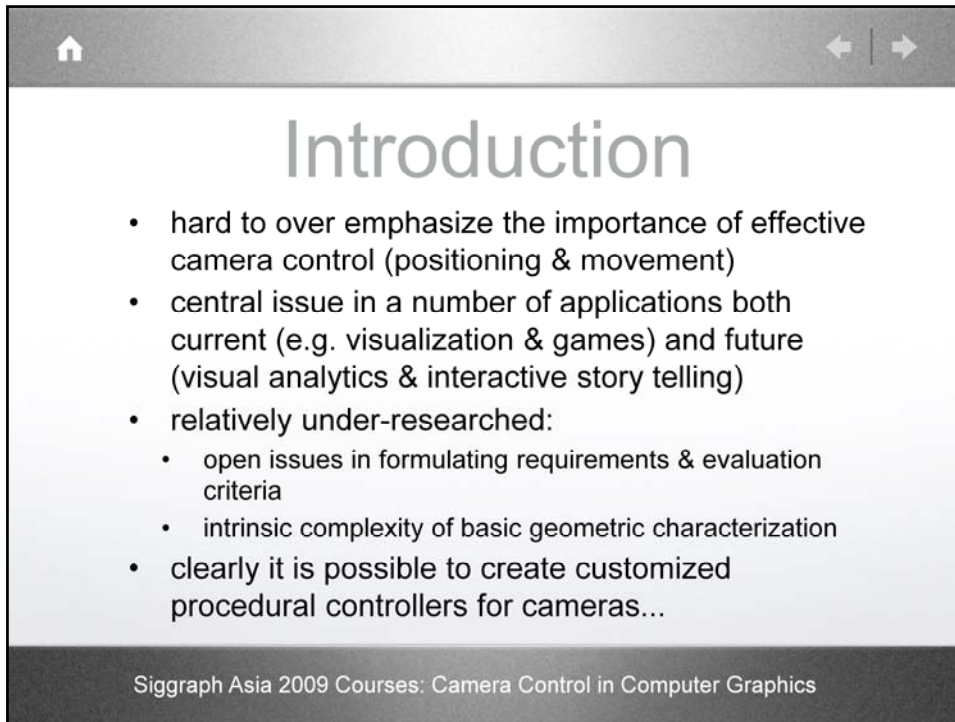
Chapter 3

Annotated Slides



Self-introduction to the presenters and their backgrounds:

- Marc Christie (INRIA, France)
- Patrick Olivier (Newcastle University, UK)



The slide is titled "Introduction" and features a list of bullet points. The slide has a dark header bar with a home icon on the left and navigation arrows on the right. The footer bar contains the text "Siggraph Asia 2009 Courses: Camera Control in Computer Graphics".

- hard to over emphasize the importance of effective camera control (positioning & movement)
- central issue in a number of applications both current (e.g. visualization & games) and future (visual analytics & interactive story telling)
- relatively under-researched:
 - open issues in formulating requirements & evaluation criteria
 - intrinsic complexity of basic geometric characterization
- clearly it is possible to create customized procedural controllers for cameras...

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

From an applications perspective, approaches to camera control can be distinguished on the basis of whether the user exercises some degree of interactive control, or the application assumes full control of the camera itself. Interactive approaches propose a set of mappings between the dimensions of the user input device (e.g. mouse, keyboard) and the camera parameters. The nature and complexity of the mappings are highly dependent on the targeted application. Low-level approaches rely on reactive techniques borrowed from robotics and sensor planning, where the behavior of the camera is driven in direct response to visual properties in the current image. Constraint-based and optimization-based approaches reflect a move towards higher-level control in which the user specifies desired image properties for which the camera parameters and paths are computed using general purpose solving mechanisms. The range, nature and specificity of the properties characterize the expressiveness of the approach.



Course goals

1. review existing camera control schemes
2. motivate through empirical evidence:
 - application requirements
 - current practice (film and new media)
3. general camera control formulations
4. representation & reasoning requirements
5. characterize camera control research agenda

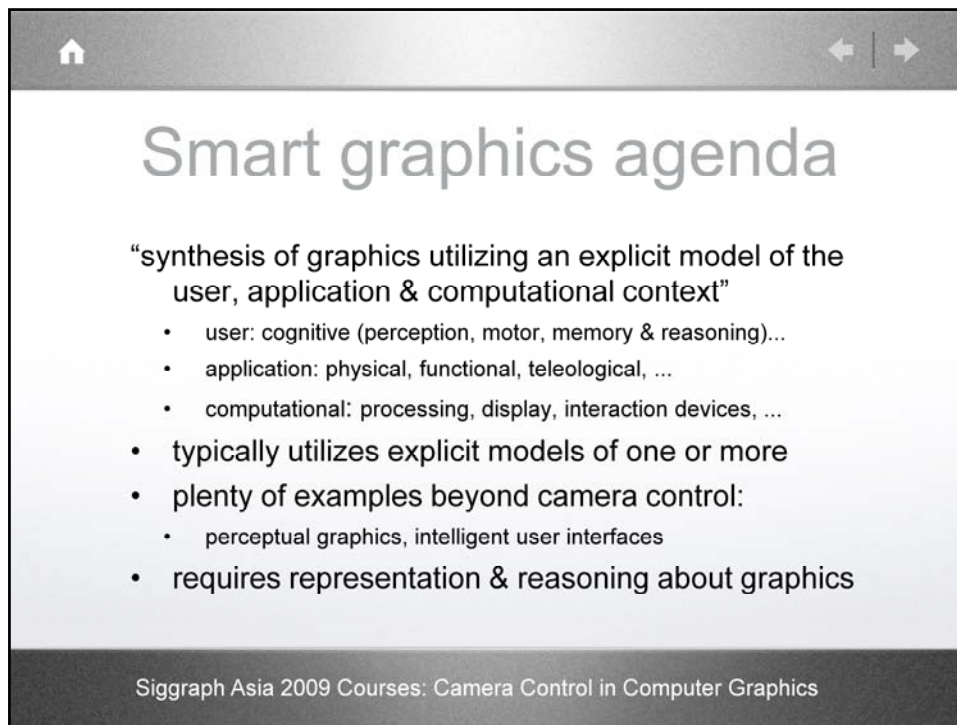
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

Broad discussion of what we expect people to get out of this and what the attendees experience of camera control has been in the past.



While most approaches have been developed in response to the specific requirements of an application domain, there are many common difficulties including the number of degrees of freedom, the computational complexity related to any path-planning problem, and the evaluation and avoidance of occlusion.

Our tutorial in camera control progresses from motivations (the applications and general cinematography practice) interactive approaches to fully automated control. After characterizing the requirements of camera control in a number of key applications, we discuss the relevance of photographic and cinematographic practice. We then consider contrasting proposals for user control of a camera and fully automated control. Throughout we emphasize the principal challenges for camera control, and conclude with a discussion of the impact of occlusion and techniques for dealing with it within different camera control formulations



The slide is titled "Smart graphics agenda" and features a list of bullet points. The slide has a header bar with a home icon and navigation arrows, and a footer bar with the course title.

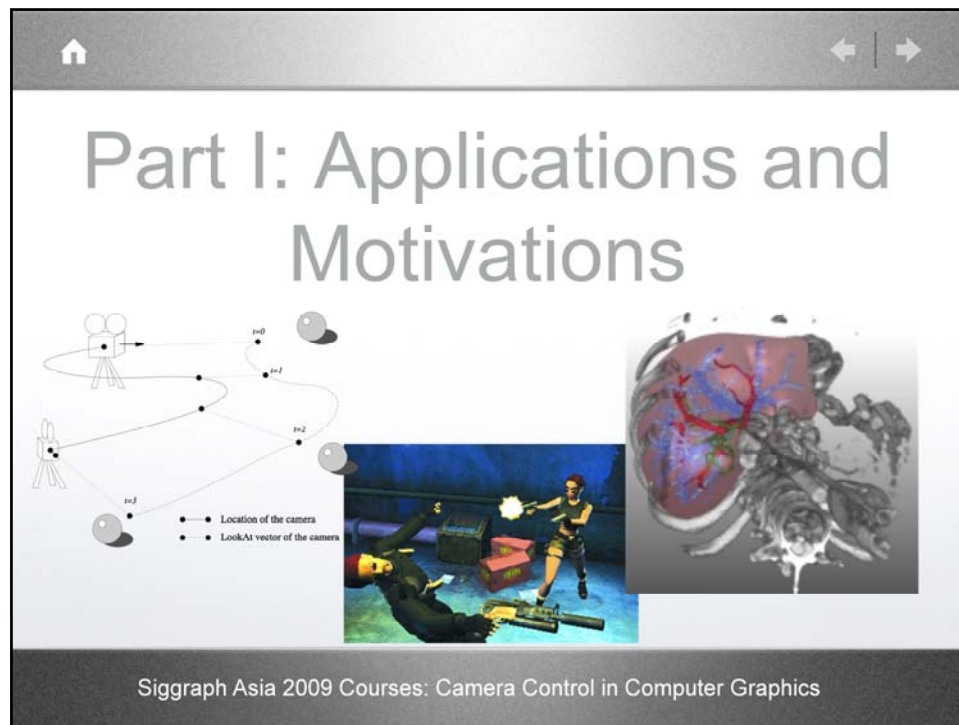
Smart graphics agenda

- “synthesis of graphics utilizing an explicit model of the user, application & computational context”
 - user: cognitive (perception, motor, memory & reasoning)...
 - application: physical, functional, teleological, ...
 - computational: processing, display, interaction devices, ...
- typically utilizes explicit models of one or more
- plenty of examples beyond camera control:
 - perceptual graphics, intelligent user interfaces
- requires representation & reasoning about graphics

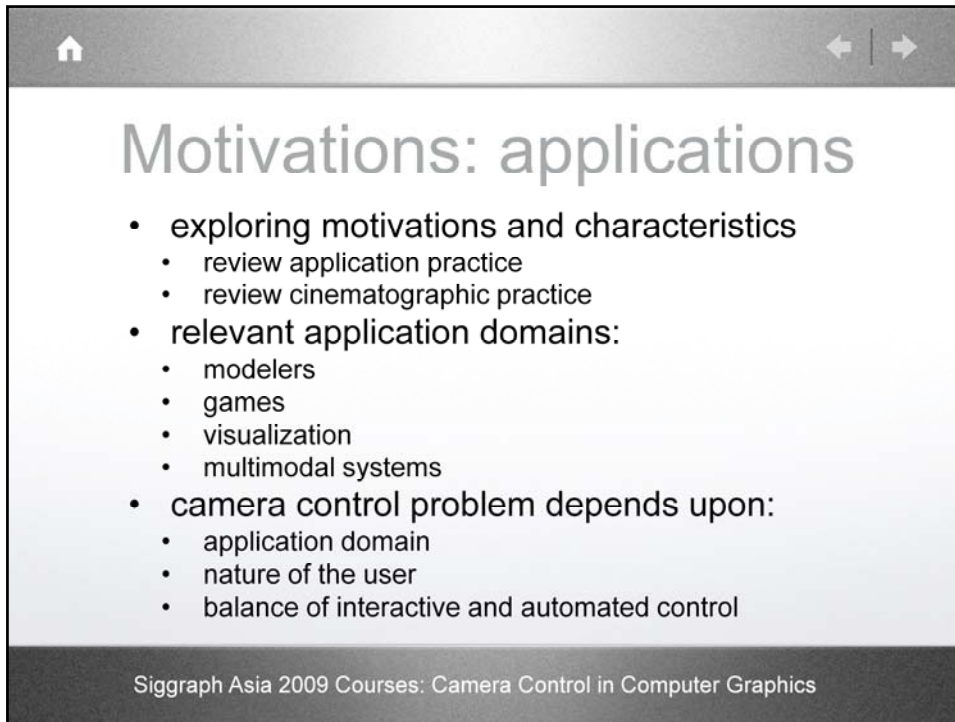
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

Smart Graphics is grounded in a deep understanding of human abilities, activities, and desires. This understanding arises through the integration of fields such as art, design, and the social, cognitive, and perceptual sciences. Insights are realized in the form of novel methods for producing and interacting with rich graphical displays often utilizing established techniques from Computer Graphics, Artificial Intelligence, and Computer Science in general. Such interfaces present content that: engages the user and is aesthetically satisfying participates in human cognition as external or distributed representations is sensitive to the real-time demands of the interaction in the context of the available computational resources and adapts the form of the output according to a wider set of constraints such as an individual's perceptual, attentive, and motor abilities and the nature of the presentation media and available interaction devices.

We understand automated and even assisted camera control within this broader intellectual vision.



From an applications perspective, approaches to camera control can be distinguished on the basis of whether the user exercises some degree of interactive control, or the application assumes full control of the camera itself. Interactive approaches propose a set of mappings between the dimensions of the user input device (e.g. mouse, keyboard) and the camera parameters. The nature and complexity of the mappings are highly dependent on the targeted application. Low-level approaches rely on reactive techniques borrowed from robotics and sensor planning, where the behavior of the camera is driven in direct response to visual properties in the current image. Constraint-based and optimization-based approaches reflect a move towards higher-level control in which the user specifies desired image properties for which the camera parameters and paths are computed using general purpose solving mechanisms. The range, nature and specificity of the properties characterize the expressiveness of the approach.



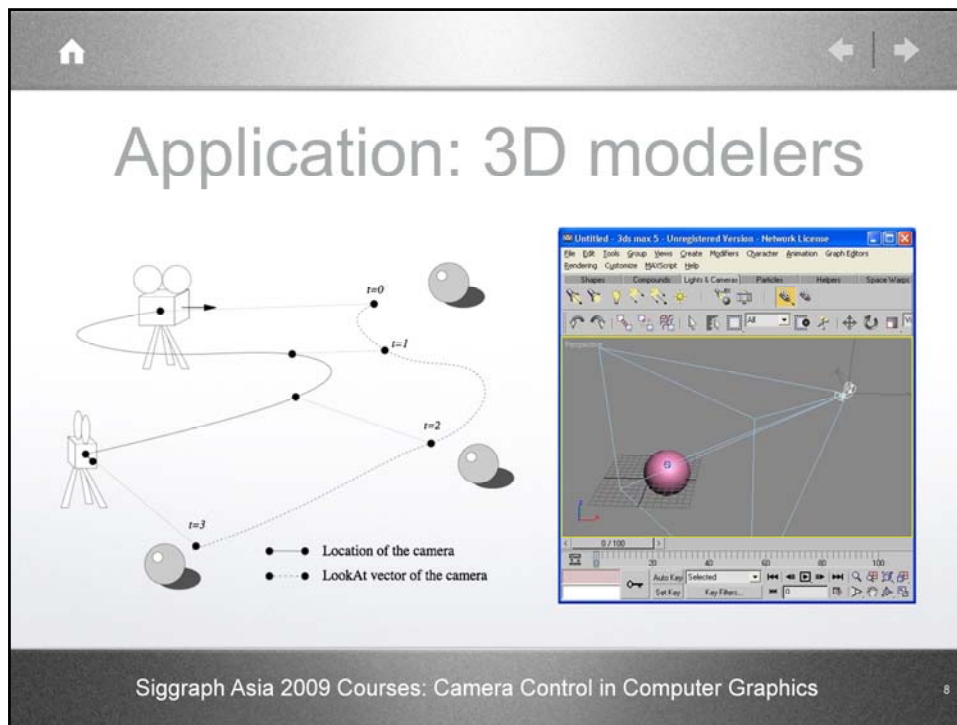
Home icon | Navigation arrows

Motivations: applications

- exploring motivations and characteristics
 - review application practice
 - review cinematographic practice
- relevant application domains:
 - modelers
 - games
 - visualization
 - multimodal systems
- camera control problem depends upon:
 - application domain
 - nature of the user
 - balance of interactive and automated control

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

The requirements for interactive and automated approaches to camera control can in part be found in the use and control of cameras in a number of common computer graphics applications. Though existing applications are bound by the state-of-the art in camera control itself, the goals of the user and existing use of interactive and automated control, provide us useful insights as to the needs of future applications.

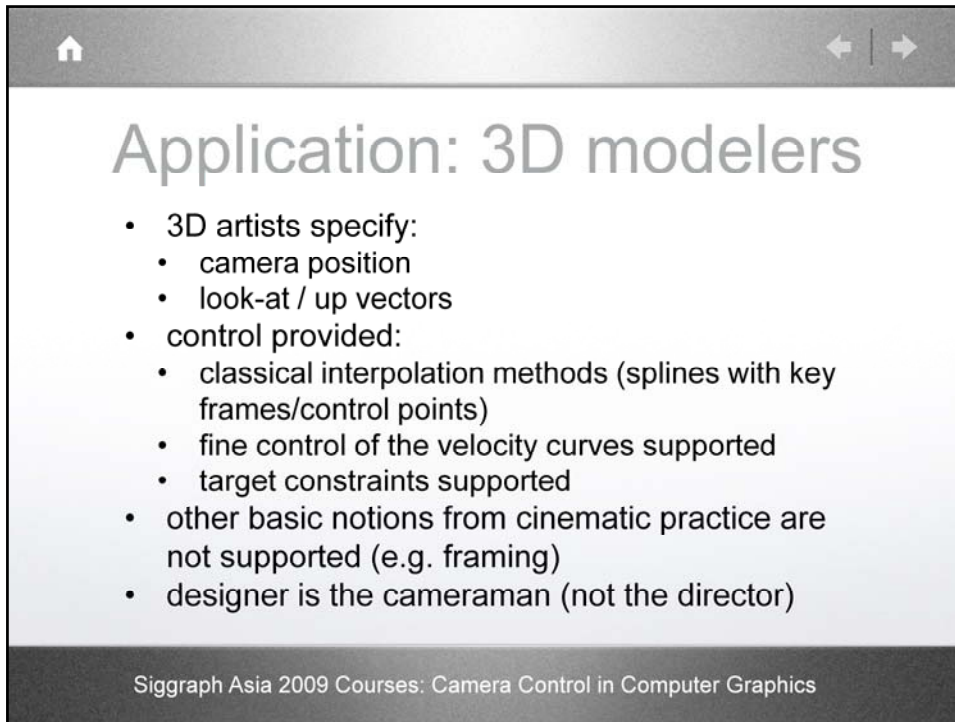


In three-dimensional modeling environments, virtual cameras are typically configured through the specification of the location of the camera and two vectors that represent the look-at and up directions of the camera. The specification of camera motion is usually undertaken through a combination of direct editing and interpolation, such as the use of splines with key frames and/or control points. Animation of the camera is realized by interpolating the camera location, up and look-at vectors across key frames. Fine control of camera speed is provided through the ability to manipulate the velocity graphs for each curve.

A set of complementary tools provides modelers with the ability to use the position of a unique static or dynamic target object to constrain the look-at vector. Modelers may also allow the use of offset parameters to shift the camera a small amount from the targeted object or path. Similarly, some tools allow constraints to be added to fix each component of the look-at vector individually. Physical metaphors are also used to aid tracking, such as virtual rods that link the camera to a target object. With the possibility to extend the functionality of modelers through scripting languages and plug-ins, new controllers for cameras can be readily implemented (e.g. using physics-based systems). Furthermore, with the rise of image-based rendering, the creation of camera paths using imported sensor data from real cameras is increasingly popular.



Video of camera control in a 3D modeling tool – illustrating the principal controls.



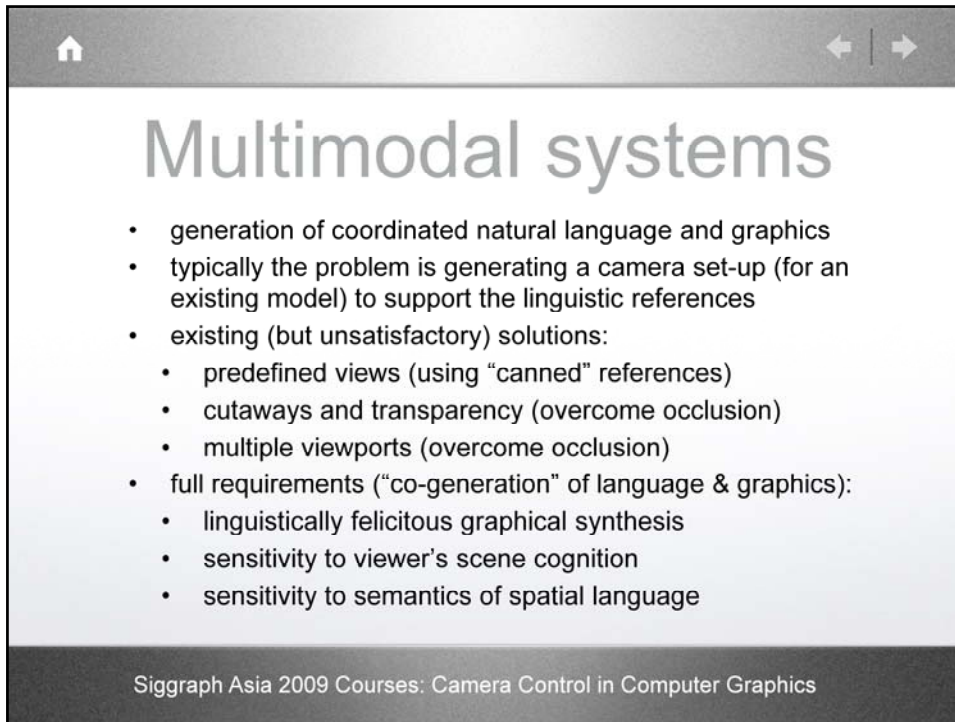
Application: 3D modelers

- 3D artists specify:
 - camera position
 - look-at / up vectors
- control provided:
 - classical interpolation methods (splines with key frames/control points)
 - fine control of the velocity curves supported
 - target constraints supported
- other basic notions from cinematic practice are not supported (e.g. framing)
- designer is the cameraman (not the director)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

In practice, the underlying camera control model (i.e. two spline curves) is not well suited to describing the behavioral characteristics of a real world cameraman, or the mechanical properties of real camera systems. Despite the fact that a number of proposals exist for describing cinematic practice in terms of camera position, orientation and movement, most modelers have not attempted to explicitly incorporate such notions in their tools. Even basic functionality, such as automatically moving to an unoccluded view of a focal object, cannot be found in current commercial modeling environments.

This mismatch can in part be explained by the general utility that most modeling environments strive to achieve. Cinematic terminology is largely derived from character oriented shot compositions, such as *over-the-shoulder shots*, *close shots* and *mid shots*. *Operating in these terms* would require the semantic (rather than just geometric) representation of objects. Furthermore, the problem of translating most cinematographic notions into controllers is non-trivial, for example, even the seemingly simple notion of a *shot* will encompass a large set of possible, and often distinct, solutions. However, providing users with high-level tools based on cinematic constructs for the specification of cameras and camera paths, would represent a significant advance over the existing key-frame and velocity graph-based controls.





The slide is titled "Multimodal systems" in a large, bold, sans-serif font. It features a list of bullet points discussing the generation of coordinated natural language and graphics, the problem of generating camera set-ups, and existing solutions like predefined views and cutaways. The footer text is "Siggraph Asia 2009 Courses: Camera Control in Computer Graphics".

- generation of coordinated natural language and graphics
- typically the problem is generating a camera set-up (for an existing model) to support the linguistic references
- existing (but unsatisfactory) solutions:
 - predefined views (using "canned" references)
 - cutaways and transparency (overcome occlusion)
 - multiple viewports (overcome occlusion)
- full requirements ("co-generation" of language & graphics):
 - linguistically felicitous graphical synthesis
 - sensitivity to viewer's scene cognition
 - sensitivity to semantics of spatial language

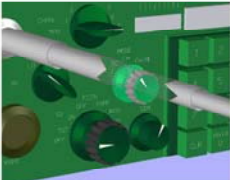
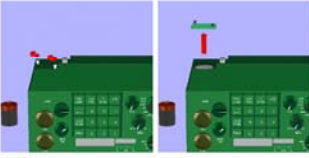
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

The generation of multimodal output (e.g. natural language and graphics) involves careful coordination of the component modalities. Typically such systems have been developed in the domain of education and training and in particular need to address the problem of coordinating the choice of vantage point from which to display the objects being described, or referred to, linguistically. For example, a direct linguistic reference to an object (e.g. the handle on the door) usually requires that the object (i.e. the handle) is no more than partially occluded in the shot. To satisfy such coordination constraints, multimodal generation systems have relied heavily on the use of default viewpoints from which unoccluded views of the elements of discourse are likely to be achieved. Ray casting is used to trivially accept or reject viewpoints although attempts have been made to address the application of constraint-based camera planning in the development of a prototype intelligent multimedia tutorial system. Alternative approaches use cutaways and ghosting, standard devices in engineering graphics, by which occluding elements of scene are removed either by direct surgery on the polygons, manipulation of the depth buffer or object transparency.

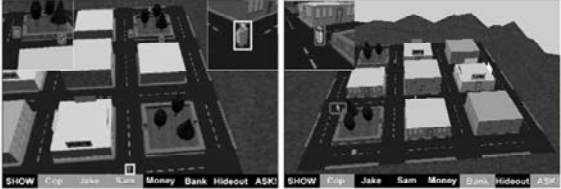


Multimodal systems

Remove the old holding battery Step 1 of 2



Step 1:
Remove the holding battery cover plate, highlighted in the right picture:
Loosen the captive screws and pull the holding battery cover plate off the radio.



- default views
- transparency

Seligmann & Feiner [SF91,SF93]

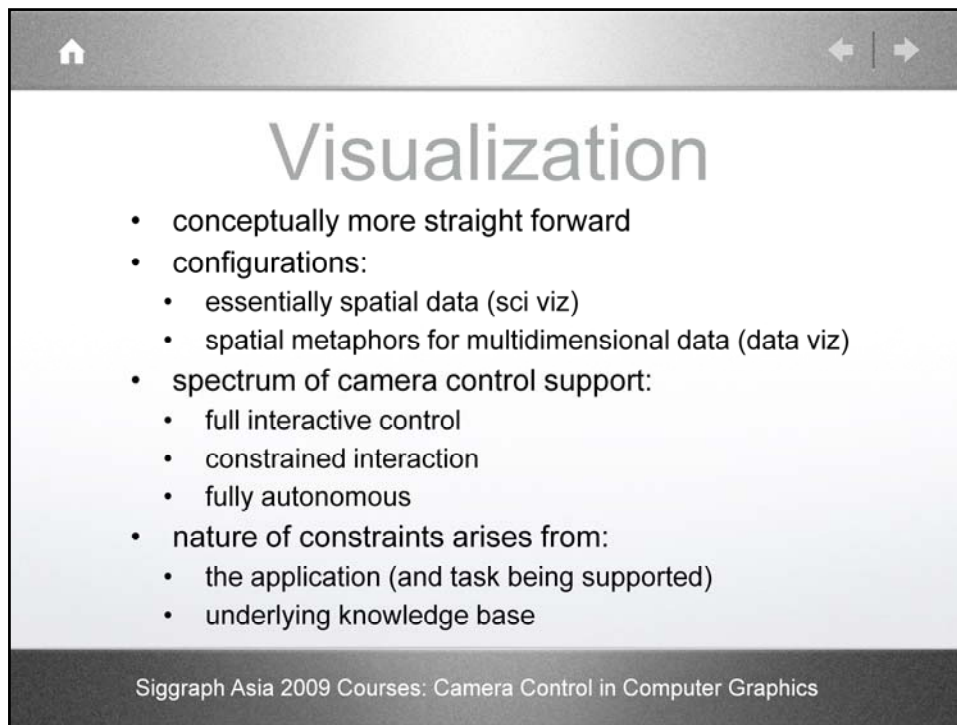
- multiple views

Bares & Lester [BL99]

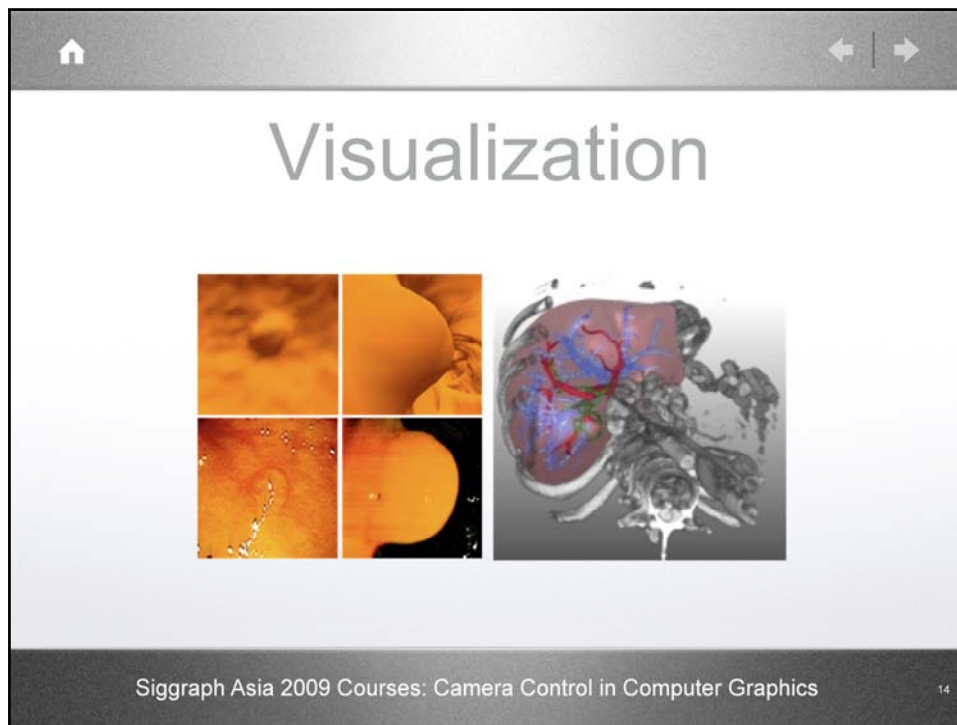
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

12



Beyond simple object references, the coordination of language and graphics poses a number of interesting problems for camera control. Indeed, such applications are a rich source of constraints on a camera, as the semantics of some spatial terms can only be interpreted by reference to an appropriate perspective. For example, descriptions involving spatial prepositions (e.g. in front of , left of) and dimensional adjectives (e.g. big, wide) assume a particular vantage point. For projective prepositions the choice of a deictic or intrinsic reference frame, for example, for the interpretation of in front, directly depends on the viewpoint of a hypothetical viewer.



In visualization systems, multidimensional data sets may be mapped to different three-dimensional spatial entities with a view to furnishing users with an intuitive and interactive framework to explore the underlying relations. Typically, such data sets, and the resulting visualizations, are often vast landscapes of geometry within which manual interactive control is extremely difficult. Visualization is an application for which the user requires interactive control to explore and pursue hypotheses concerning the data. However, user interaction in such applications is usually restricted to a small number of navigational idioms, for example, the identification of a number of *interesting points* or regions in the data, and the exploration of the remaining data in relation to these. Automatic camera control and assisted direct camera control, has the potential to greatly enhance interaction with large data sets.





In practice, even partially automated three-dimensional multimedia generation requires an interpretation and synthesis framework by which both the visuospatial properties of a viewpoint can be computed (i.e. the interpretive framework) and the viewpoint controlled according to the constraints arising from the semantics of the language used (i.e. the synthesis framework). Likewise, future scientific and information visualization systems will benefit greatly from intelligent camera control algorithms that are sensitive to both the underlying characteristics of the domain and the task that the user is engaged in. Such adaptive behavior presupposes the ability to evaluate the perceptual characteristics of a viewpoint on a scene and the capability to modify it in a manner that is beneficial to the user.



Application: Games



- require camera control
 - during game play (real-time)
 - between game play (cut scenes)
- available resources tightly constrained
- classes of viewpoint
 - first person
 - third person
 - bird's eye
 - cinematic
- key problems:
 - occlusion vs geometric complexity
 - gameplay vs cinematic qualities
 - visual consistency



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 15

Interactive computer games serve the benchmark application for camera control techniques. Most importantly, they impose the necessity for real-time camera control. A canonical camera control problem involves following one or more characters whilst simultaneously avoiding occlusions in a highly cluttered environment. Furthermore, narrative aspects of real-time games can be supported by judicious choice of shot edits both during and between periods of actual game play. The increasing geometric complexity of games means that most deployed camera control algorithms in real-time 3D games rely upon fast (but fundamentally limited) heuristic occlusion checking techniques, such as ray casting (see later for a discussion of occlusion).

Camera control in games has received considerably less attention in computer games than visual realism, though as John Giers (a game developer at Pandemic Studios) noted, “the camera is the window through which the player interacts with the simulated world” [Gio04]. Recent console game releases demonstrate an increasing desire to enhance the portrayal of narrative aspects of games and furnish players with a more cinematic experience. This requires the operationalization of the rules and conventions of cinematography. This is particularly relevant in the case of games that are produced as a film spin-offs, where mirroring the choices of the director is an important means of relating the game play to the original cinematic experience.



Application: Games

"Problems such as clunky controls and a frustrating camera, which were excusable in the early games, have steadily degraded the quality of the series releases over time.... Once you get the hang of it you can effortlessly overcome even the most imposing obstacles without difficulty. It's also always abundantly clear which ledges you can hang on or jump between, so the only challenge is positioning the camera so you can see where you're trying to go, which can be frustrating. In tight spots it can be difficult to get a good view of the ledge you need to jump to, and sometimes it's easy to misjudge a jump if you don't have the camera aligned just right. The camera problems are intermittent though, and most of the time you have a fairly good view of the surroundings."

http://shopper.cnet.com/Tomb_Raider_Legend_PC/4512-9696_9-31568589.html



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics16

- **First person**

Users control the camera (giving them a sense of being the character in virtual environment). Many games use first person camera views, and the most common genre is the First Person Shooter (FPS), for example, the Doom and Quake series. Camera control is unproblematic, since it is directly mapped to the location and orientation of the character.

- **Third person**

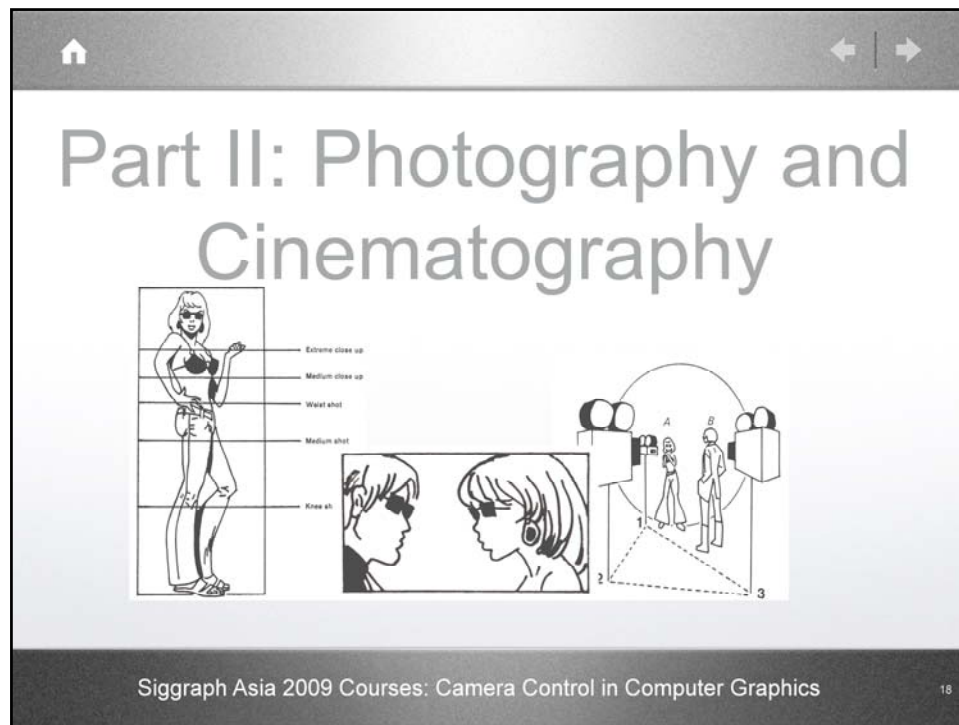
The camera system tracks characters from a distance, generally the view is slightly above and behind the main character) and responds to both local elements of the environment (to avoid occlusion) and the character's interactions (maintaining points of interest in shot). Problems arise when the shot fails to support important events in the game, for example, when a character backs-up against a wall such systems typically default to a frontal view, thereby disrupting the game play by effectively hiding the activity of opponents. Furthermore, due to the imprecise nature of the occlusion detection procedures in game camera systems (e.g. ray casting), partial but often significant, occlusion of the main character is a common occurrence.

- **Action replay**

Replays are widely used in modern racing or multi-character games where there are significant events that a player might like to review. It is imperative that these replays are meaningful to the players, such that the elements of the scene and their spatial configuration are readily identifiable. Interactive storytelling presents a number of interesting opportunities for camera control. In particular, the explicit representation of both narrative elements and character roles, relations and emotional states, provides a rich basis on which to select shots and edits automatically.

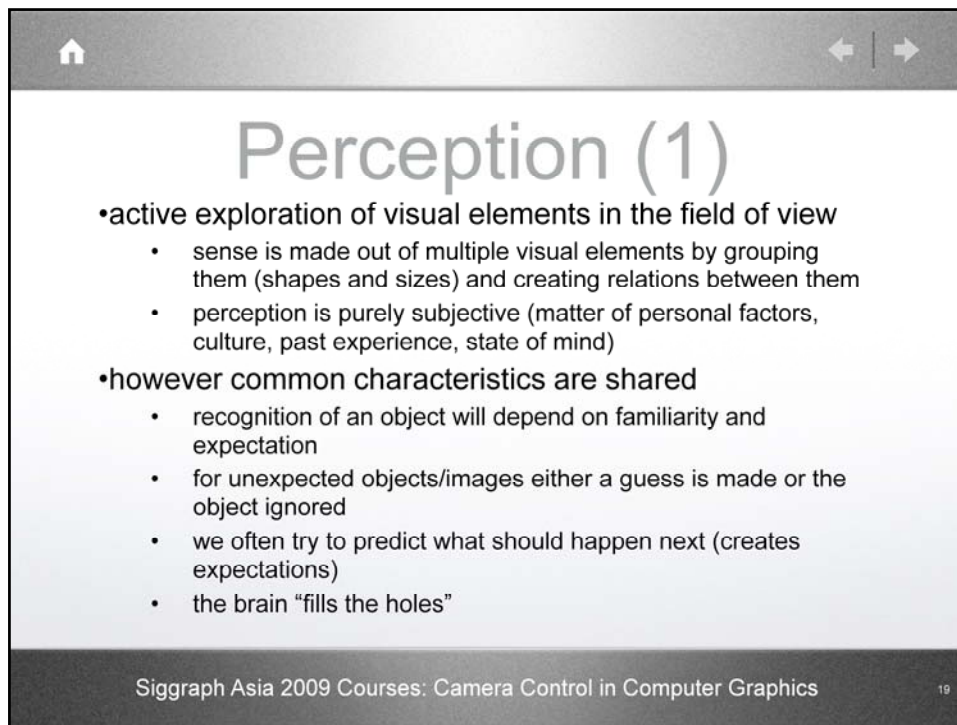


Games are inherently different from film in that the camera is usually either directly or indirectly controlled by players (typically through their control of characters to which a camera is associated). Furthermore, a game is a dynamic and real-time environment and game camera systems must be responsive to action that takes place beyond the focal characters. The enforcement of frame coherency (smooth changes in camera location and orientation) is necessary to avoid disorienting players. While the automation of camera control based on cinematographic principles aims to present meaningful shots, the use of editing techniques (which are rare) can preserve game play by presenting jump-cuts or cut-scenes to guide the user. The use of automated editing and cinematographic techniques in games is currently the exception rather than the rule.



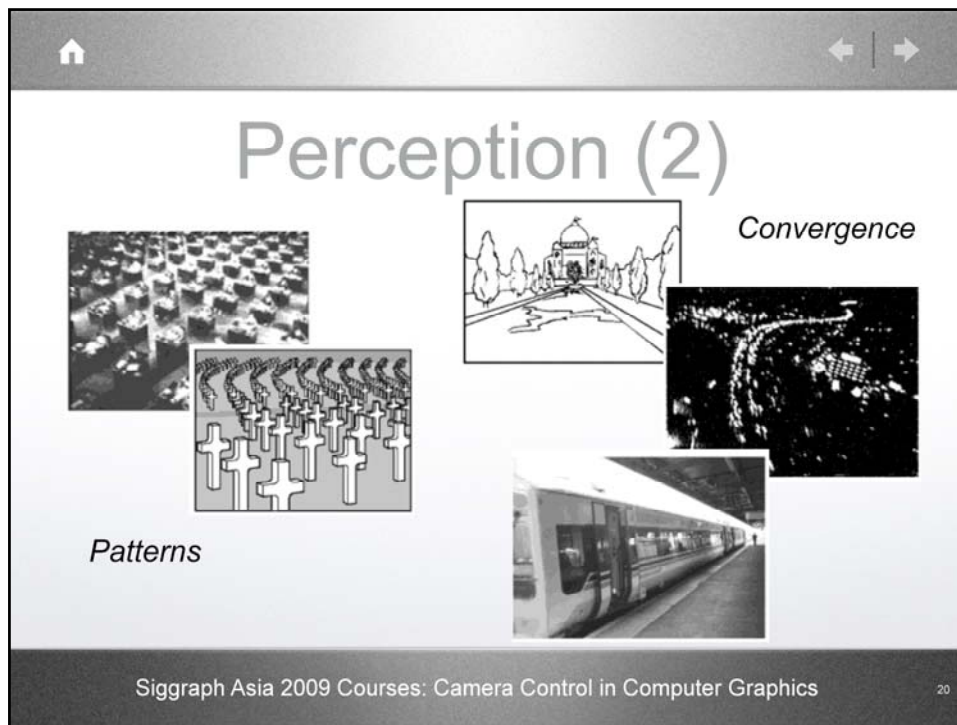
This section will mainly provide an overview of photography and cinematography techniques. We will explore fundamentals of picture perception and techniques to properly layout elements to create pleasing, balanced or dissonant views. We will show how these principles are applied to still shots e.g. for conveying narrative elements (eg. dominance relations between characters, creating oppression,...). We will then detail some sets of rules and conventions founded in cinema to convey actions and narrative elements, for a broad range of actors configurations and movements, and explore more fundamental principles such as spatial continuity, continuity of action and of motion.

We will finally discuss the psychological aspects subtending pictures and sequence of pictures.



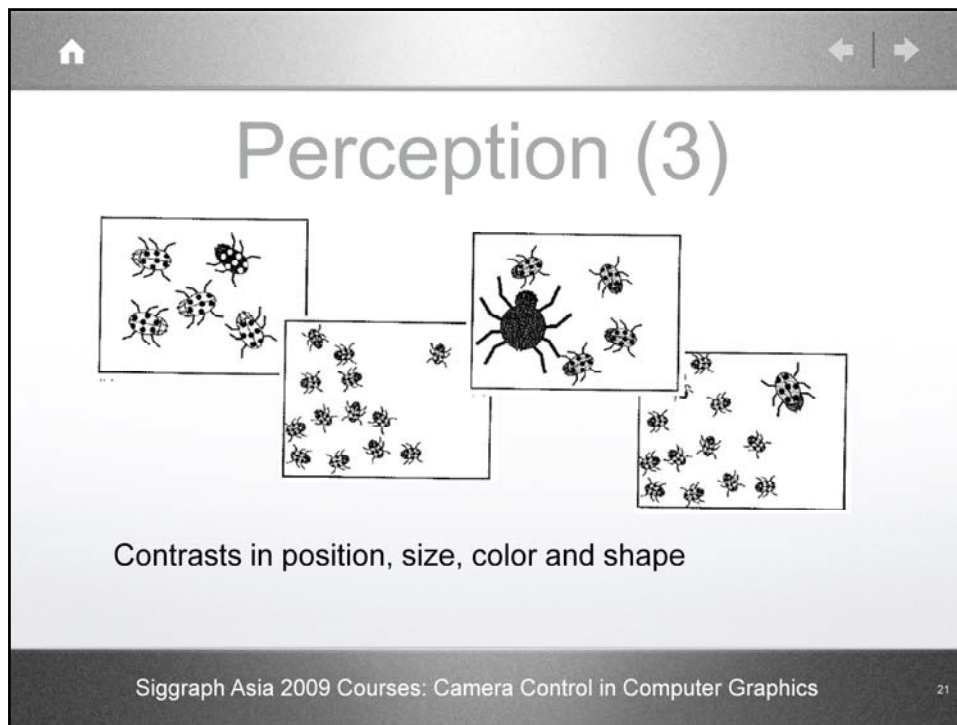
Perception has been mainly influenced by Gestalt theories. These claim that we perceive a general view, the entire image, rather than isolated elements (we always need to group elements, either by shape or by size -- see pictures on next slide)

One can interestingly analyse camera shots and camera motions with these theories and furthermore (try to) rely on them to build intended effects (though filmmakers adopt a more practical approach).

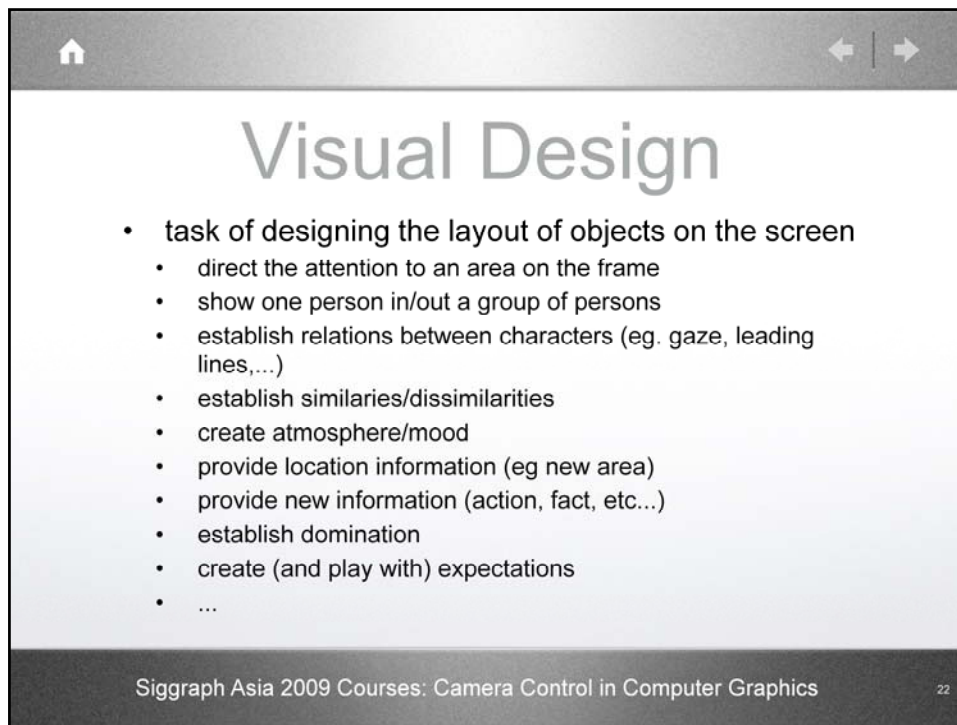


Important perceptual characteristics (for composition) encompass:

- grouping and organization (one rapidly builds a clustering of shapes, and identifies groups compared to isolated elements – see next slide)
- balance: visual weight of an element or group of elements on the screen wrt the whole. Balance helps to emphasize the important elements, as well as establish relations (eg. equity, dominance) between elements, as well as moods (balanced vs. unbalanced shots).
- shapes: primitive forms such as leading lines, circles or curves. Convergence helps to guide the viewer towards a target of interest, while setting up the relation between the target and his environment (eg. the train and the character in the bottom picture). One generally avoids lines that divide the frame into equal parts (a fundamental of the rule of the thirds/fifths), especially wrt. Horizon in outdoor environments. Lines help to structure the attention, and rely either on geometric elements of the picture (direct or with closure gestalt), or through invisible elements (direction of gaze, links between multiple faces in a picture)
- Patterns: repetition of geometric figures. Using a regular repetition (windows of a building with no perspective) gives a static view, whereas repetition with a perspective gives a sense of dynamism (see left picture) .

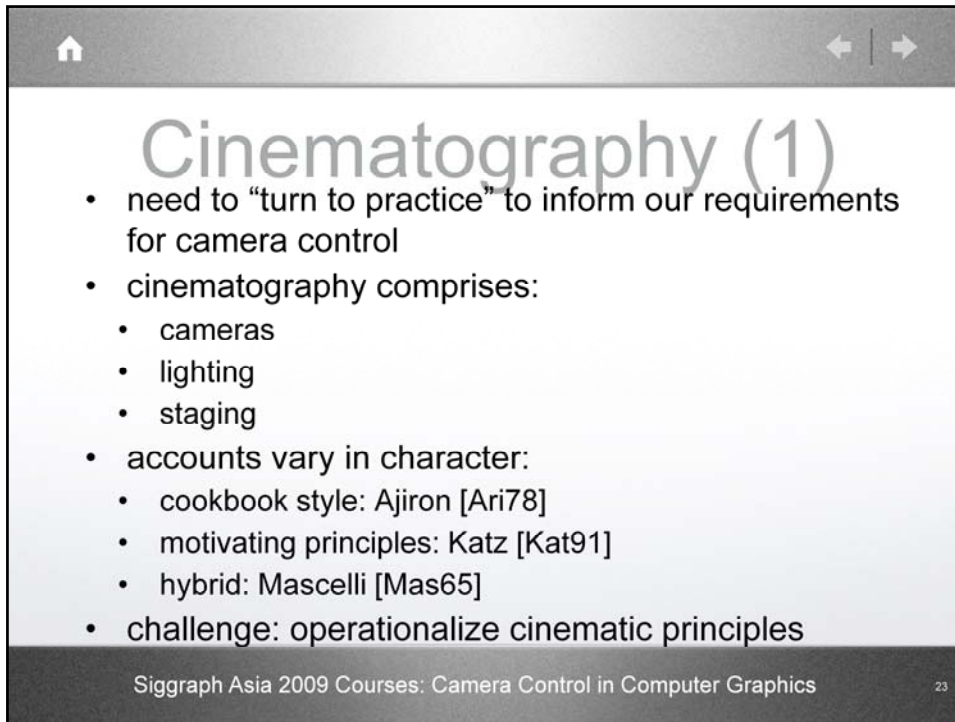


Contrast (a variation of illuminance) is one of the first visual feature we perceive in a picture, and helps to establish relations and groups. Here we refer to contrasts in terms of position (distance between elements), size, colors and shapes. Connecting stars of the zodiac is one of the first illustration of contrasts in position. In cinematography and photography, relative locations of characters in crowds are regularly used to express isolation or integration.



Shots and sequences of shots are motivated by communicative goals. Emphasizing these goals requires to better guide the attention of the audience, by structuring the narrative elements in each shot, over the shots (and between the shots). Visual design consists in organizing the elements on the screen to draw the audience attention and fulfill the communicative goal. Two examples are:

- providing new information: since people always connect successive shots as a whole, new (and important facts) to the narrative need to be emphasized (eg using convergence, leading lines, ...)
- creating expectations: further in the process, at a cognitive level, people try always to complete the story by guessing the hidden facts, creating and trying to check their hypothesis. This creates expectations, that one can play with by hiding/displaying expected/unexpected facts. An unbalanced fact, for example can create some form of spatial expectation (meaning there may be another character in the scene, but not seen on the screen)



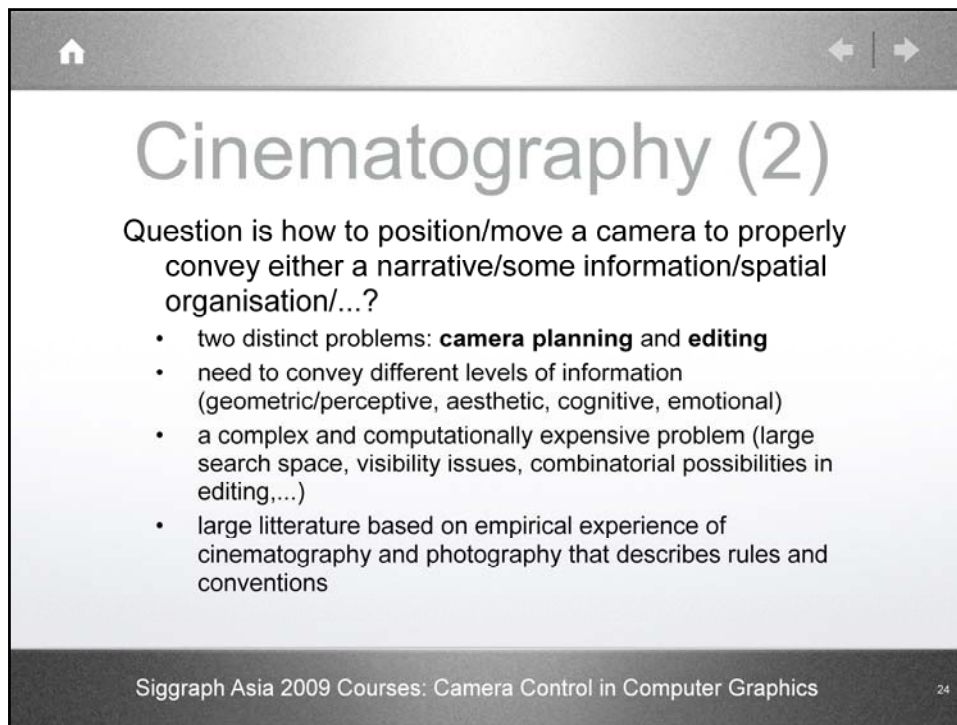
Cinematography (1)

- need to “turn to practice” to inform our requirements for camera control
- cinematography comprises:
 - cameras
 - lighting
 - staging
- accounts vary in character:
 - cookbook style: Ajiron [Ari78]
 - motivating principles: Katz [Kat91]
 - hybrid: Mascelli [Mas65]
- challenge: operationalize cinematic principles

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 23

There is a considerable consensus as to the nature of best practice in cinematography. Accounts such as Arijon classify the sets of possible viewpoints, and sequences of viewpoints according to the number of actors and their location in the environment. Such accounts are apparent in a number of automated cinematography systems that rely on finite state machines to operationalize the editing principles, for the specific case of dialogues.

Higher-level motivations are more difficult to formalize. Mascelli describes cinematography in broader motivation principles (narrative, spatial and temporal continuity).



There are multiple aspects to consider when operationalizing rules from the literature. First the problem is actually two-fold: we distinguish the problem of viewpoint (and motion) planning, from the problem of editing (when to cut a shot, and where to move to).

Second, both a single shot and a sequence of shots convey different levels of information:

- geometric: convey the content of the screen (ie what are the objects on the screen, where are they located)
- aesthetic: convey balance, unity, by the way elements are organized on the screen (leading lines, visual weights, ...)
- cognitive: convey the understanding of elements in the narrative (facts, spatial knowledge of the environment, relations between characters, causal effects, ...)
- emotional: convey by multiple means (music, composition, cuts, mental state,...) the emotional content.

Cinematography (3)

Though a film can be considered to be nothing but a linear sequence of frames, it is mandatory to think of its design in a structured way:

- a film is a sequence of *scenes* (each scene captures some specific situations and actions)
- each scene is composed of a sequence of *shots* (camera location/motion with temporal and spatial continuity)
- shots are separated by *cuts* (cuts are cognitively unnoticed, and audiences are "well educated")

Cinematography offers:

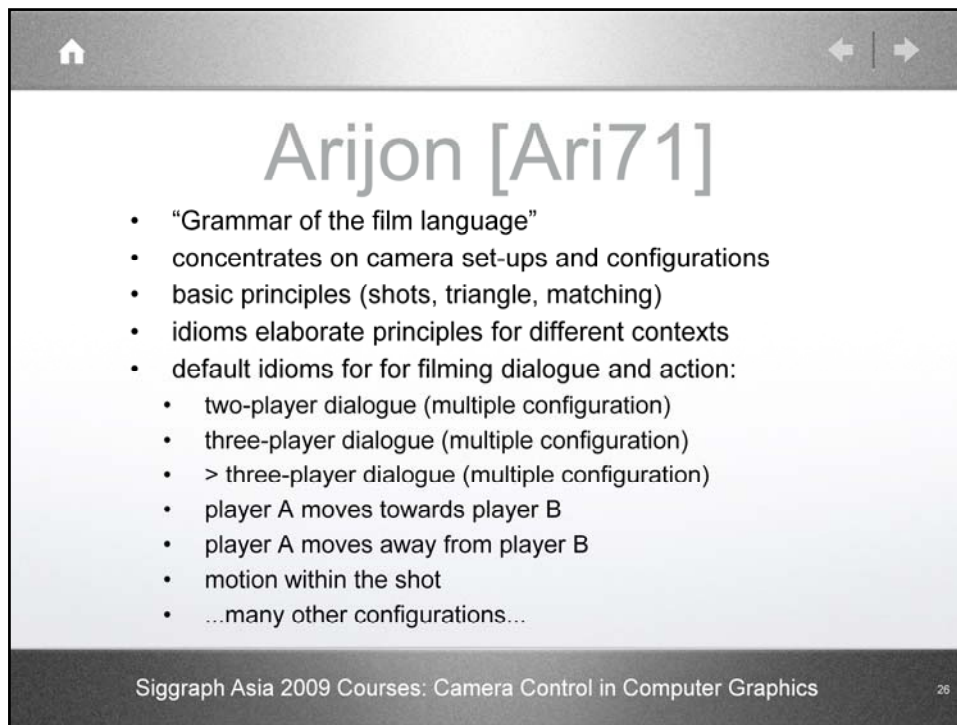
- general purpose rules (shots, triangle principle, matching)
- typical ways of shooting actions (e.g. dialogue) that are called *idioms*

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 25

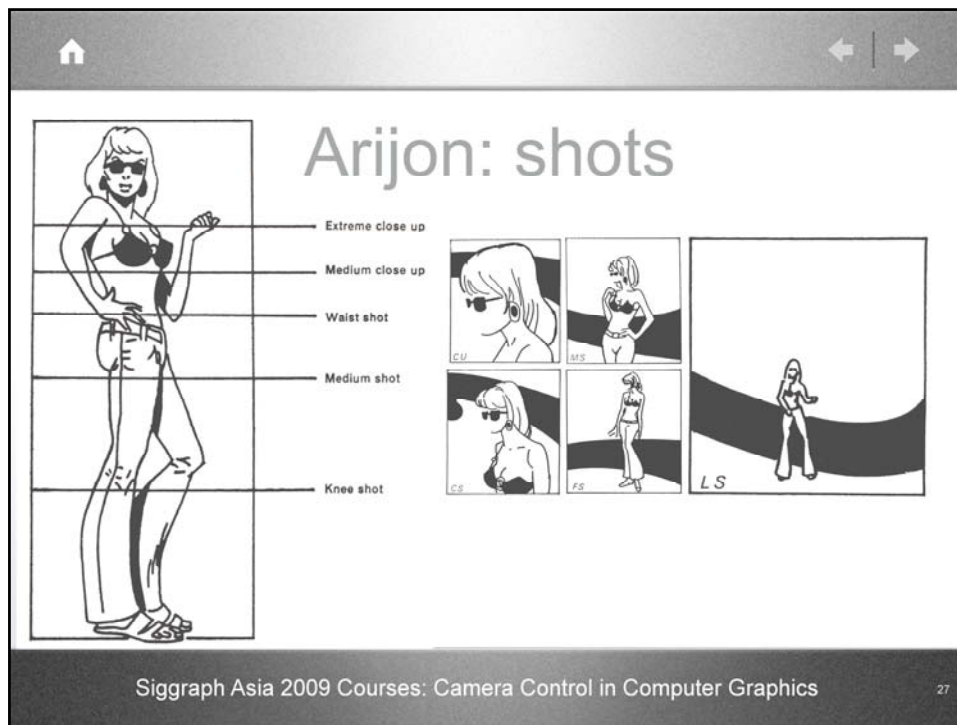
Narrative structure of a film. We distinguish the level of the story (the set of actions, the causal relations between actions) and the level of the discourse (how to convey these actions or narrative elements).

The discourse level structures a film into scenes (a scene follows some form of continuity in the action and in the space), each scene is composed of shots, separated by cuts. Techniques can be used to interleave scenes (parallel editing), to complete some partial knowledge, to illustrate competing actions in time, etc.

How and when to perform cuts in a movie remains a critical issue (and has consequently received little attention in its operationalization): inappropriate cuts are very noticeable in movies (since audiences are well educated). And audience naturally establishes relations between successive shots (see Kuleshov's experience).

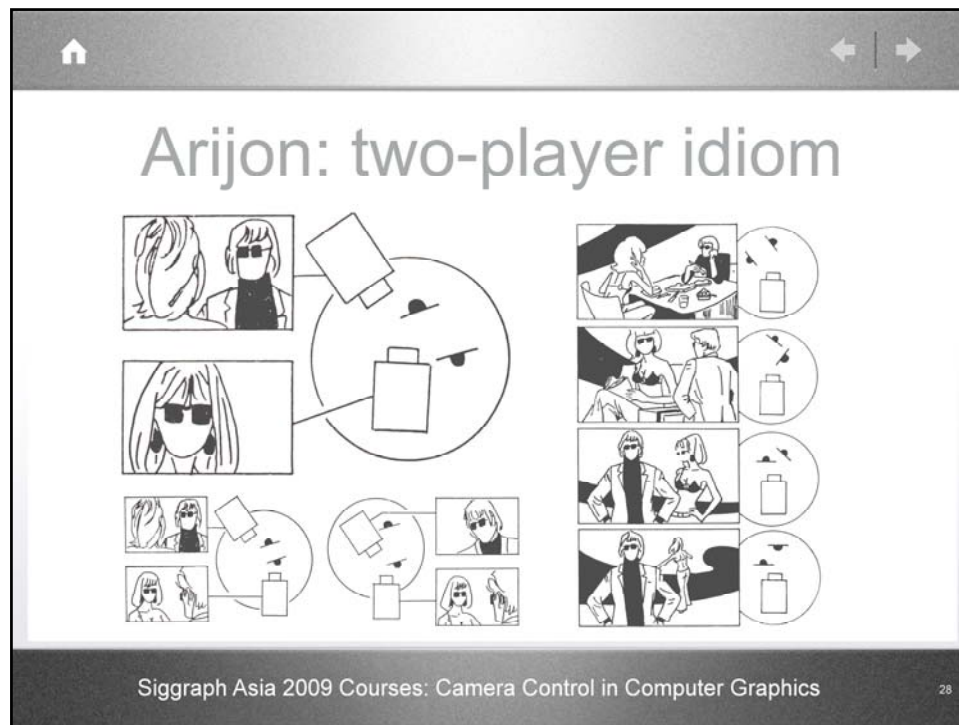


Arijon provides a detailed description of shots and cuts for typical staging of actions, in many configurations (referred as cinematographic idioms). The next slides will illustrate both common idioms, and idioms to be used in specific action sets.



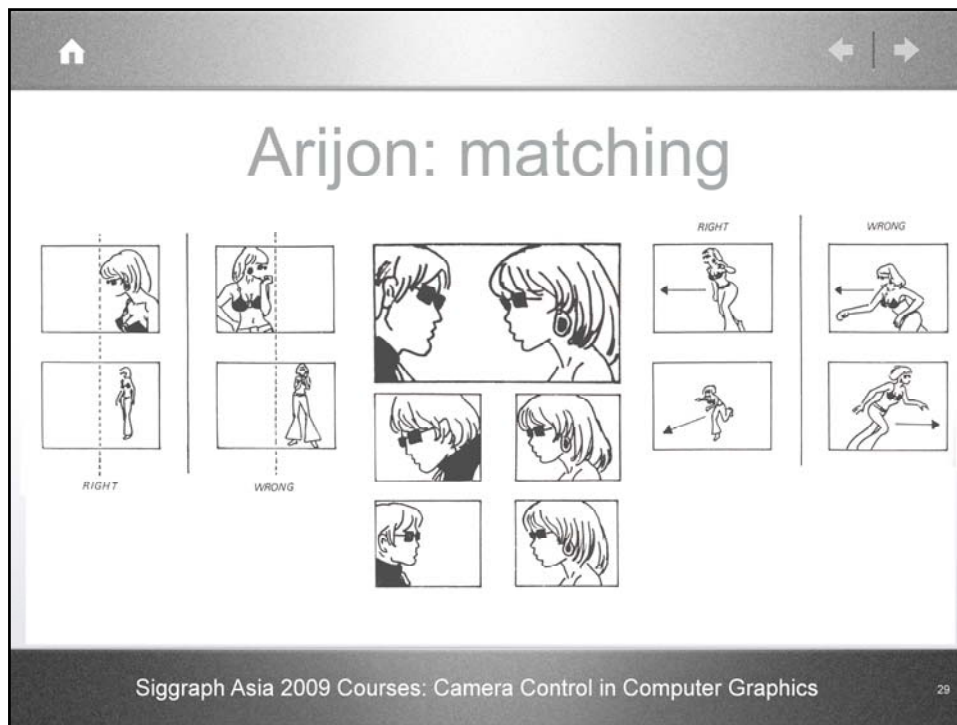
Only a subset of shots are appropriate for shooting a character. These are characterized by the amount of the subject in the screen.

- long shots display the character in relation to its environment or to other characters (eg. for establishing shots)
- full shots frame the character fully on the screen. FS are used to track moving actors.
- medium shots are classically used in dialogues (we still see the expressive parts of the body: hand, shoulders, head)
- close shots are mainly for dialogue, illustrates the body posture as shoulders are kept in frame
- Medium close up displays mostly the characters face (capture emotions, follow the speech)
- Extreme close ups are used to emphasize the characters reaction (face, eyes) or attitude.



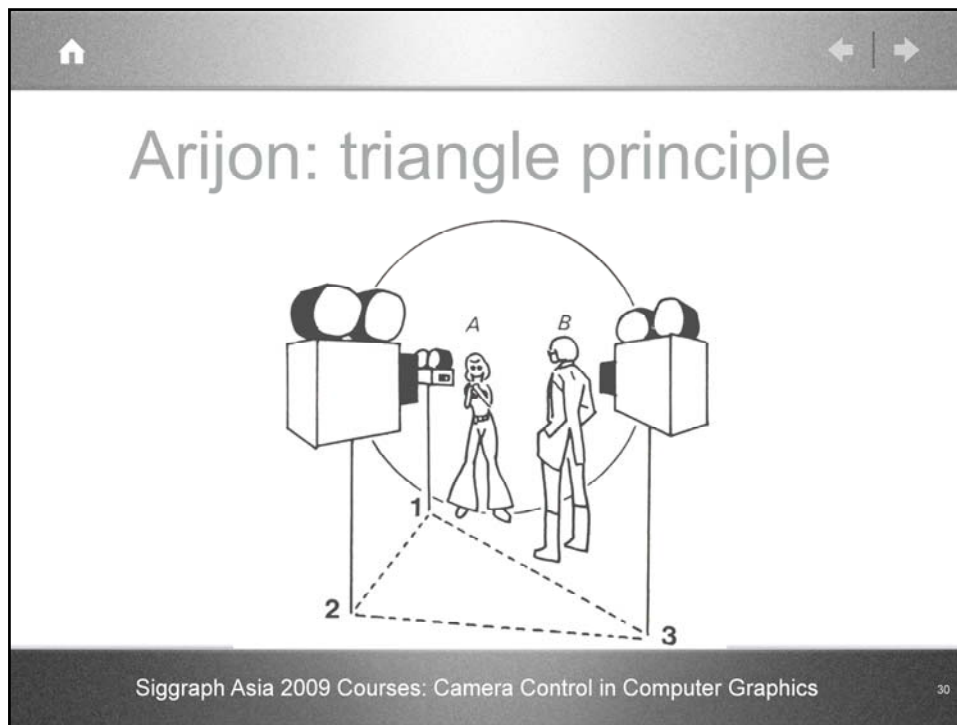
Typical camera sets (idioms) for conversation. A virtual line is drawn between the actors (the line of interest), and all shots are located on one side of this line. Then multiple camera viewpoints can be employed: apex shots (we see both characters, same size on screen), external shots – or over-the-shoulder shots, internal shots, where only one character is framed, and subjective shots in which the camera represents the eyes of an actor.

Depending on the stylistic choice of editing, cuts between shots can be more or less frequent (pace) and dependent on the actors utterances (switching before each change, or maintaining same viewpoint on exchanges).



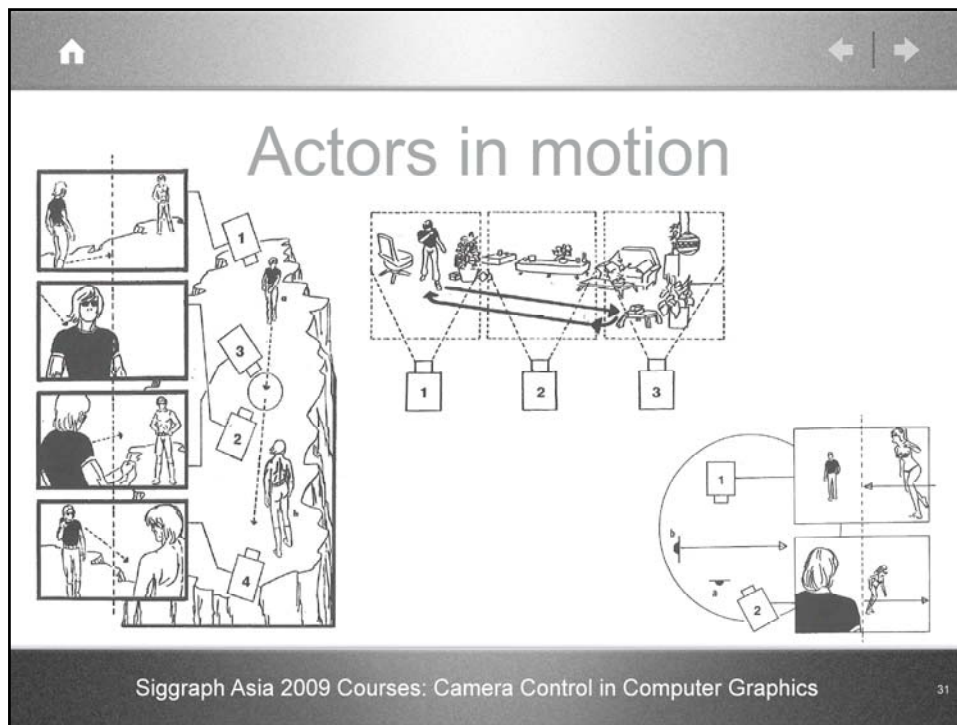
There are many rules to maintain some form of continuity between successive shots:

- relative placement (ensures spatial continuity): a character framed on one side of the shot should be maintained on that side for the duration of the scene. This is modulated by the actions that occur in the scene, the number of actors involved, and the possibility to cross the line of interest in specific configurations (occlusion, continuous transition between two viewpoints)
- maintain line of interest while framing a dialogue (spatial continuity)
- maintain relative motion of the character (action continuity). Changing the direction of the actor on the screen will deliver an impression of running in different directions as if the character was lost.
- avoid jump cuts: jump cuts occur when the difference between two successive shots is not strong enough (ie difference in orientation lower than 30 degrees, or no change in the size of the shot)



Triangle principle. 1 and 3 draw the line of interest, and 2 is the location of the apex view. Shots are internal (1 character framed) or external (2 characters framed). Parallel shots frame one of the character from a direction orthogonal to the line of interest.

External shots (over the shoulder shots) generally give a sense of identification with the closest character.



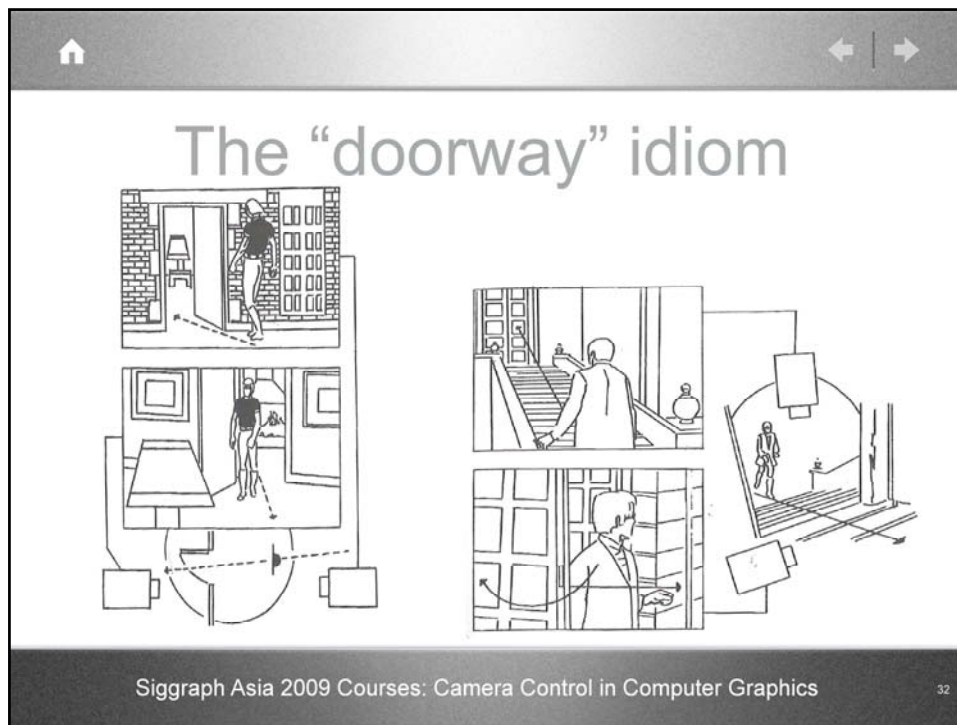
Multiple examples of actors in motion:

- on the left picture: character (a) is moving towards character (b) illustrated by four successive shots:

- 1: external full shot showing (a) moving to (b)
- 2: internal reverse close shot showing (a)
- 3: external close shot showing (a) and (b)
- 3: external reverse shot showing (a) and (b)

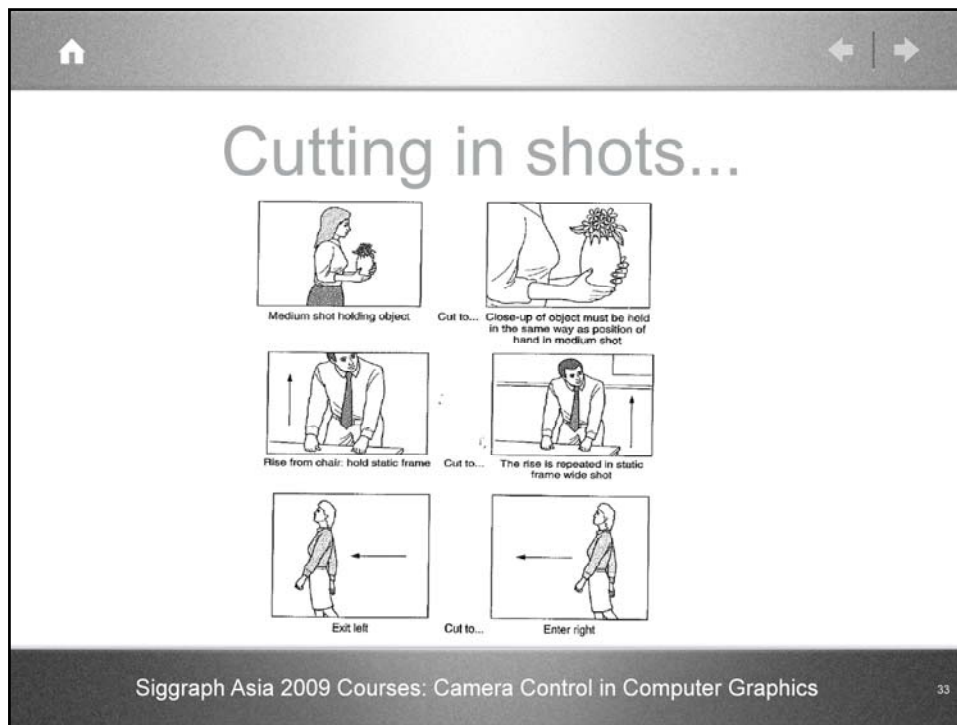
- on the middle picture: three shots that show how the character is moving in the environment. As the character is turning, in the third shot, is it important to frame the action of turning and cut on this action to illustrate continuity.

- on the right picture: two shots demonstrating character (b) passing in front of (a). Interesting setup, since though the line of action is broken between the shots (b is changing direction in successive shots), spatial continuity in picture maintains the coherency from the audience point of view: character (a) is maintained as a landmark to avoid disruption.



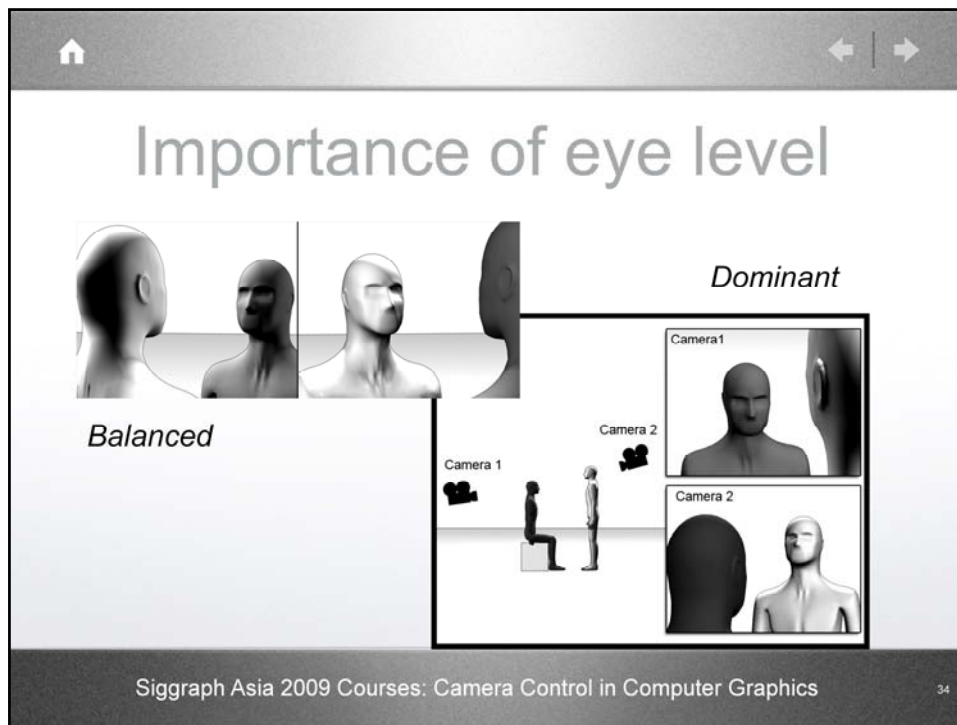
Idioms specific to the doorway action: conveying a character that is crossing entering/leaving a room:

- on the left: the relative directions is maintained between both shots (camera lays the same side of the action). An ellipsis can be used here to compress the time (ie not following the real-time evolution of actions by cutting obvious/uninteresting aspects): the audience will reconstruct the events (especially when as simple as crossing the doorway). In an opposite way, the filmmaker can overlap shots in time (showing the character opening the door in the first shot, and showing him performing the same action with a slight overlap in the second) to insist on the importance of the action (which may create here some sense of suspense).
- on the right: interestingly the line of action is violated between the two scenes. Yet spatial continuity is still ensured by sharing common objects in the scene (here the door that is recognizable in both shots).



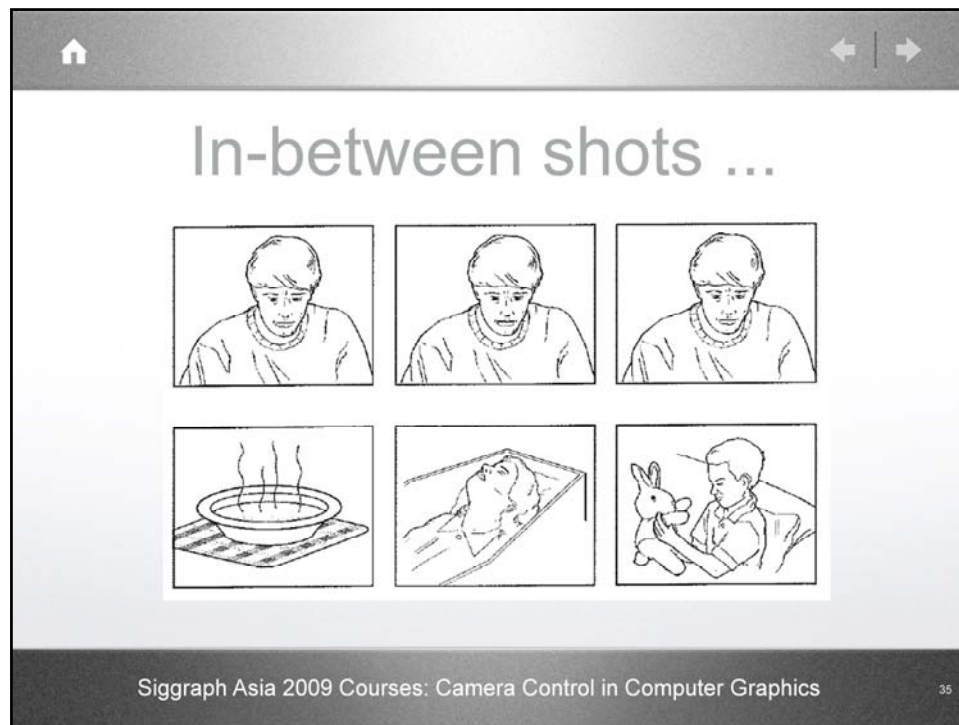
The audience should not be aware of the cuts performed by the filmmaker (the best editor is the one that is not noticeable). Actions are generally used to support the cuts in edits. Three examples of cuts in action are displayed:

- at the top: for a continuous action, cuts can be performed by a change of size in the shot (and no change in orientation)
- at the middle: for a short term action, cut is performed by a change of size, generally at the first third of the duration of the action (heuristic). The action should be displayed in both shots, which maintains the continuity.
- at the bottom: for a continuous action, the cut is performed when the character leaves the frame.



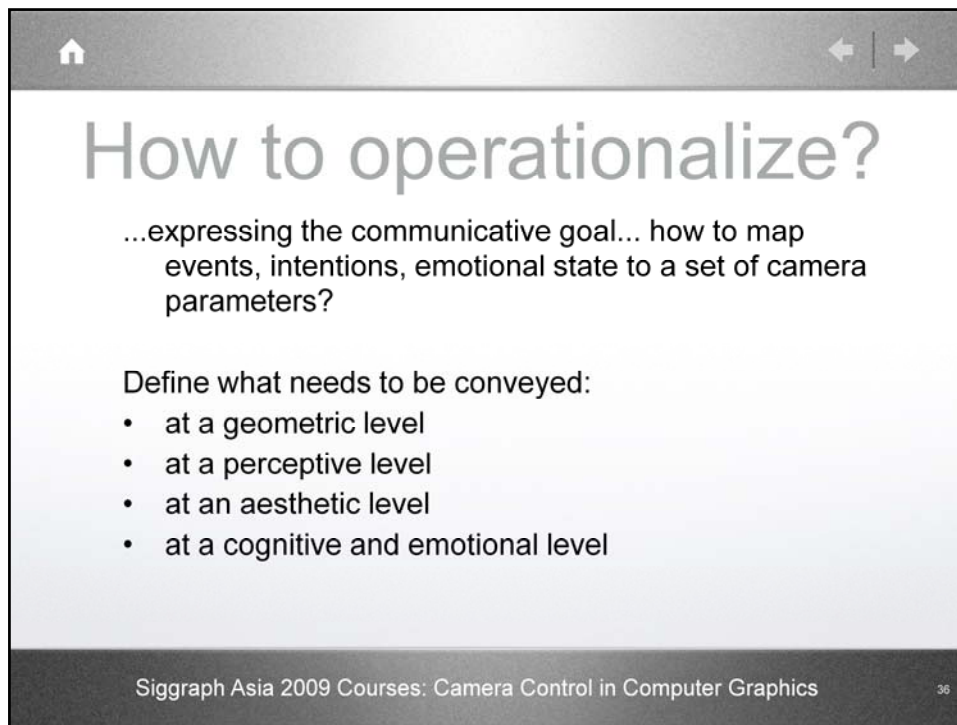
In dialogues, the height of eyes is a noticeable device to establish relationships between characters. Eyes at the same level deliver a sense of balance between the characters and supports the narrative in that way. Unbalanced shots where gazes are at significantly different levels (see right picture) gives a sense of dominance – which can be emphasized with low-angle/high angle shots.

Continuity (especially between internal shots) can be maintained by ensuring the constant height of the gaze in successive shots (shot/reverse shot).

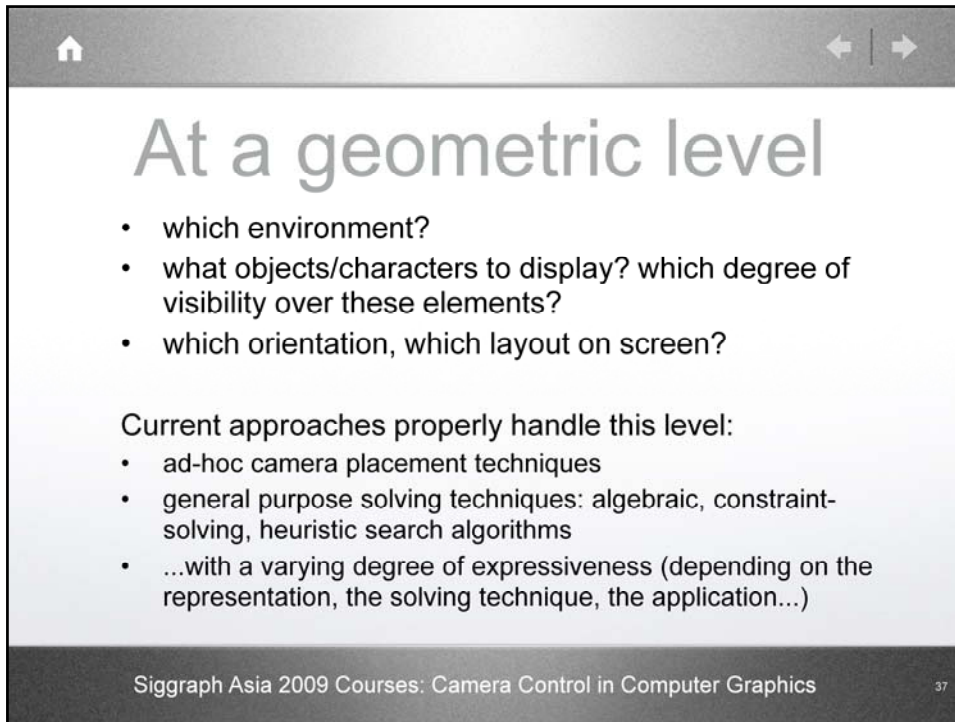


Example of Kuleshov's experiment (1939): establishing relationships between shots. No shot can be considered in isolation: sequences must be considered in their context. This can be interestingly used to misguide the audience by letting them build false relationships between facts. Fade-ins and fade-outs help to cut between scenes (and avoid erroneous interpretations), using a closure Gestalt (fade-out finishes the sequence).

This draws attention to the need of displaying only appropriate shots (if a shot is in isolation, audience will build hypothesis on connections). A scene (and hence a film) must be viewed as a global structured construction of how the audiences mind and knowledge should evolve in time.



After this short presentation of techniques employed in both photography and cinematography, we draw some comments on means and techniques to convey the four levels of information contained in images/movies (though the levels in a number of cases easily overlap).



The slide is titled "At a geometric level" and contains a bulleted list of questions and a section titled "Current approaches properly handle this level:" followed by another bulleted list. The slide has a dark header with a home icon and navigation arrows, and a dark footer with the course name and slide number.

At a geometric level

- which environment?
- what objects/characters to display? which degree of visibility over these elements?
- which orientation, which layout on screen?

Current approaches properly handle this level:

- ad-hoc camera placement techniques
- general purpose solving techniques: algebraic, constraint-solving, heuristic search algorithms
- ...with a varying degree of expressiveness (depending on the representation, the solving technique, the application...)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 37

The geometric level can be thought of by considering documentaries. Documentaries (should) convey a set of facts and how these facts articulate to gain a clear knowledge of the temporal and causal organisation of the story. Thus at the geometric level, main questions are:

- where does the scene take place (and how do we understand it takes place there – what are the recognizable elements)
- what the the objects and characters we need to display
- how do we organize the discourse level (order in which facts appear)

Current techniques that we review in this course handle properly the issues related to viewpoint computation (which environnement, which objects to see, which layout of elements). However, contributions to the more general structure of a narrative (decomposition into scenes) remains the exception.

At a perceptual level




Interpretation of sensory information in the visual pathway:

- recognisability of objects
- visual attention (eg moving objects...)
- spatial organization of elements in the screen
- visibility of salient features
- importance of functionality /role of objects

Requires to play with spatial organization of elements (and their motion), viewpoint computation, staging and lighting


Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 38

To operationalize properties at a perceptual level requires a proper understanding of how we perceive elements, how we organize them, recognize them and establish connections and relations. Though there are many contributions in the field of robotics and image analysis (eg . understanding a scene for a robot), few contributions only explore the importance of spatially organizing elements to achieve intended perceptual goals. Furthermore, the problem is inevitably linked with the issues in staging (related to the spatial organization) and lighting (related to the perception of contrasts), and requires to be looked at as one unique question.



At an aesthetic level?

- spatial organization of elements in the screen (play with leading lines)
- use of well-established photographic rules (of the thirds, of the fifths, relation to horizon)
- balance between masses and colors
- use of patterns
- importance of direction (converging lines)



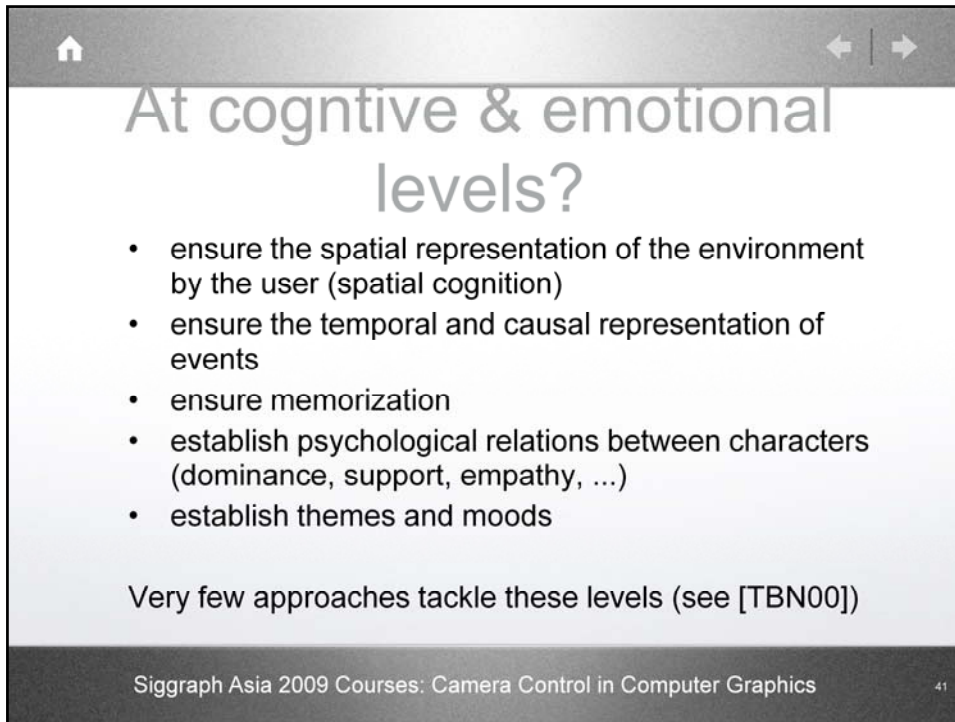
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics39

To operationalize properties at an aesthetic level requires an even deeper understanding and characterization of visual design: how to characterize balance and unity? Use of invisible leading lines and perspective lines?

Very few contributions explore such issues (see Bares et al. who change the framing to enforce composition rules, and Gooch et al. [GRMS01] who explore neighbour viewpoints to enforce e.g. the rule of the thirds through an optimisation technique).



Examples of compositions in Mascelli [Mas65], where invisible leading lines characterize the dynamism of the picture as well as the relation between elements. In the first shot, for example, the central character is obviously stuck between the others, and in a dominated position.



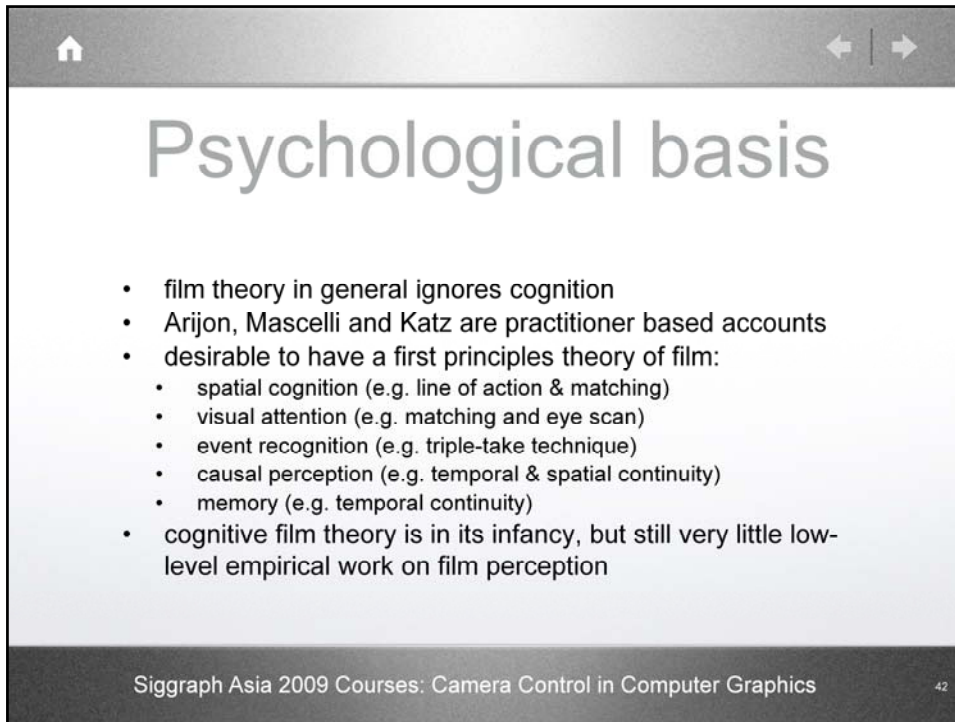
The slide is titled "At cognitive & emotional levels?". It features a list of five bullet points: "ensure the spatial representation of the environment by the user (spatial cognition)", "ensure the temporal and causal representation of events", "ensure memorization", "establish psychological relations between characters (dominance, support, empathy, ...)", and "establish themes and moods". Below the list, it states "Very few approaches tackle these levels (see [TBN00])". The footer contains "Siggraph Asia 2009 Courses: Camera Control in Computer Graphics" and the number "41".

- ensure the spatial representation of the environment by the user (spatial cognition)
- ensure the temporal and causal representation of events
- ensure memorization
- establish psychological relations between characters (dominance, support, empathy, ...)
- establish themes and moods

Very few approaches tackle these levels (see [TBN00])

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 41

At a cognitive level, the movie should provide means to properly convey the relations that subtend the actions, i.e. spatial, temporal and causal. However, apart from idioms for which we know that they properly convey such elements, there are no well founded ways (apart from empirical evidence) to operationalize cognitive and emotional aspects.

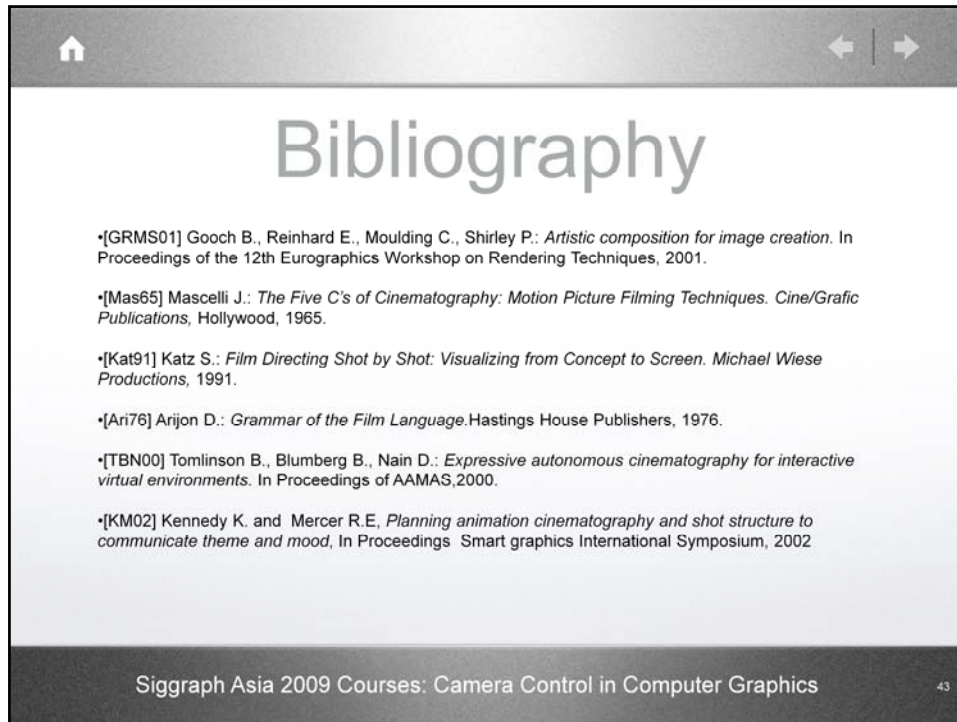


Psychological basis

- film theory in general ignores cognition
- Arijon, Mascelli and Katz are practitioner based accounts
- desirable to have a first principles theory of film:
 - spatial cognition (e.g. line of action & matching)
 - visual attention (e.g. matching and eye scan)
 - event recognition (e.g. triple-take technique)
 - causal perception (e.g. temporal & spatial continuity)
 - memory (e.g. temporal continuity)
- cognitive film theory is in its infancy, but still very little low-level empirical work on film perception

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 42

One axis of progress consists in exploring, in an empirical way, how people perceive and understand shots, as a basis to automate the computation of shots and cuts in the conveyance of a narrative.



Bibliography

- [GRMS01] Gooch B., Reinhard E., Moulding C., Shirley P.: *Artistic composition for image creation*. In Proceedings of the 12th Eurographics Workshop on Rendering Techniques, 2001.
- [Mas65] Mascelli J.: *The Five C's of Cinematography: Motion Picture Filming Techniques*. Cine/Grafic Publications, Hollywood, 1965.
- [Kat91] Katz S.: *Film Directing Shot by Shot: Visualizing from Concept to Screen*. Michael Wiese Productions, 1991.
- [Ari76] Arjion D.: *Grammar of the Film Language*. Hastings House Publishers, 1976.
- [TBN00] Tomlinson B., Blumberg B., Nain D.: *Expressive autonomous cinematography for interactive virtual environments*. In Proceedings of AAMAS, 2000.
- [KM02] Kennedy K. and Mercer R.E., *Planning animation cinematography and shot structure to communicate theme and mood*, In Proceedings Smart graphics International Symposium, 2002

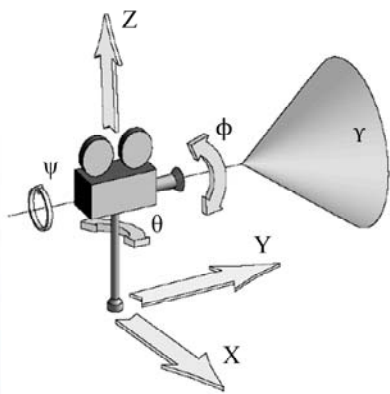
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

43



Interactive control systems modify the camera set-up in direct response to user input. The principal design issue is how to map an input device onto the camera parameters. We detail direct control techniques which rely on straightforward mappings, environment-based control that merges environment constraints (visibility, collision) in the mapping, and through-the-lens approaches that perform interaction on the screen indirectly manipulate camera parameters.

Camera parameters



ψ : roll
 ϕ : tilt
 θ : pan
Y: field of view

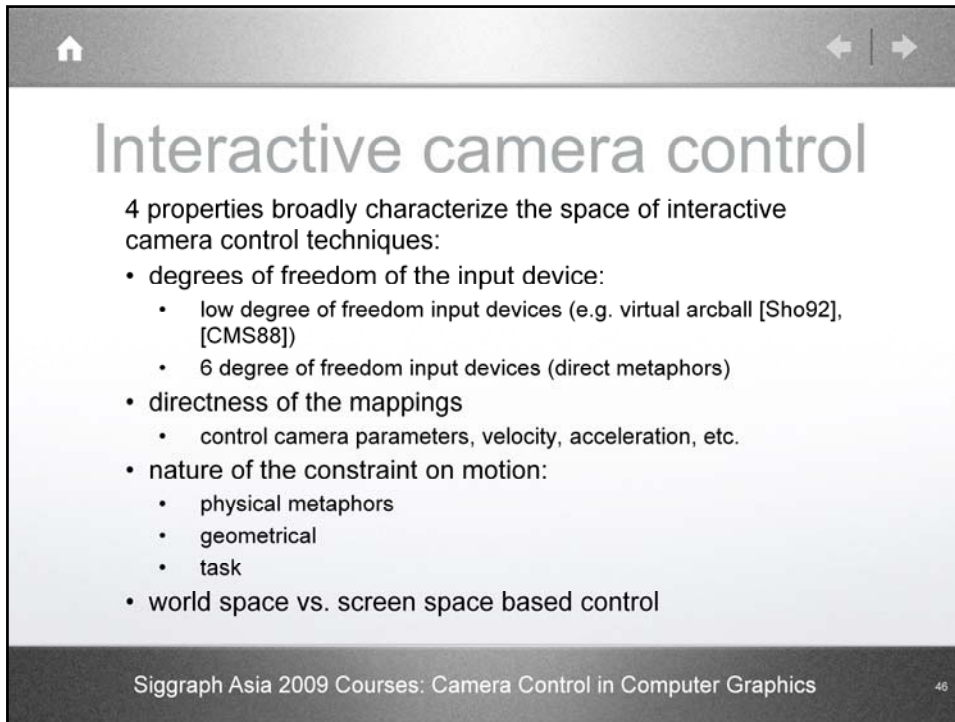
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = P(\gamma_c).R(\phi_c, \theta_c, \psi_c).T(x_c, y_c, z_c) \cdot \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix}$$
$$= H(\mathbf{q}) \cdot \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix}$$

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 45

A possible representation for the camera, with 7 degrees of freedom:

- 3 location degrees (Cartesian coordinates)
- 3 orientation degrees (Euler angles)
- 1 field of view degree

This model is well suited for direct interaction in that the orientation parameters naturally represent panoramic, tilt and roll cinematic primitives. However internal models mostly rely on quaternion-based orientation to avoid interpolation issues.



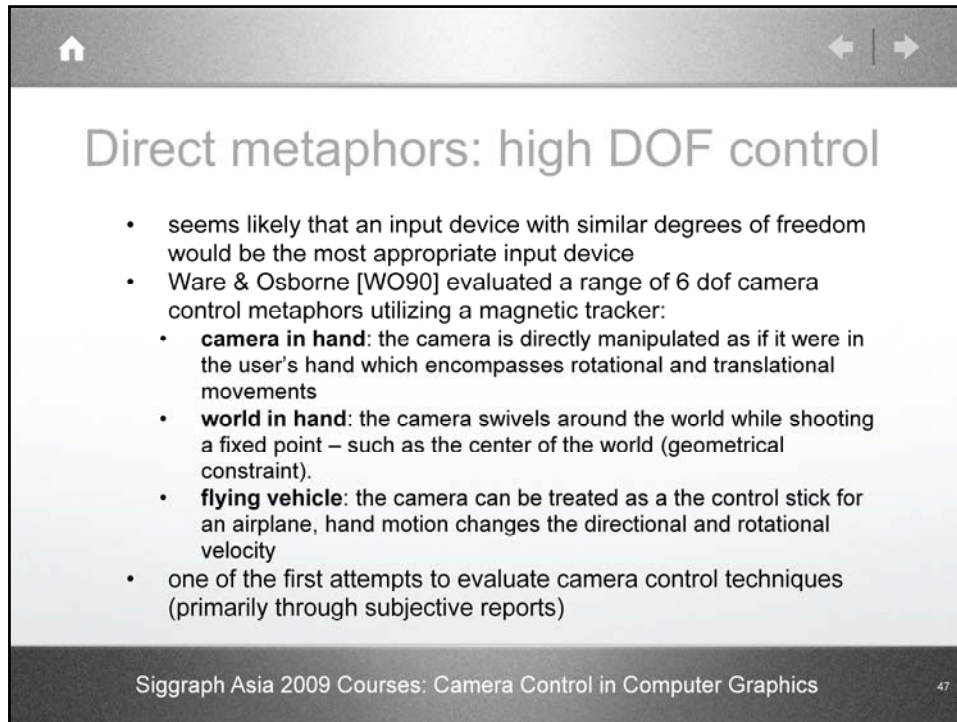
Interactive camera control

4 properties broadly characterize the space of interactive camera control techniques:

- degrees of freedom of the input device:
 - low degree of freedom input devices (e.g. virtual arcball [Sho92], [CMS88])
 - 6 degree of freedom input devices (direct metaphors)
- directness of the mappings
 - control camera parameters, velocity, acceleration, etc.
- nature of the constraint on motion:
 - physical metaphors
 - geometrical
 - task
- world space vs. screen space based control

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 46

In the domain of camera control, literature displays a large range of mappings between user inputs and camera parameters. Direct mapping techniques will associate inputs (mouse coordinates) directly to camera parameters, while indirect techniques will operate through specific interaction widgets (e.g. I-widgets [Singh06]) or spaces (screen-space [TTLCC] or application-specific space).



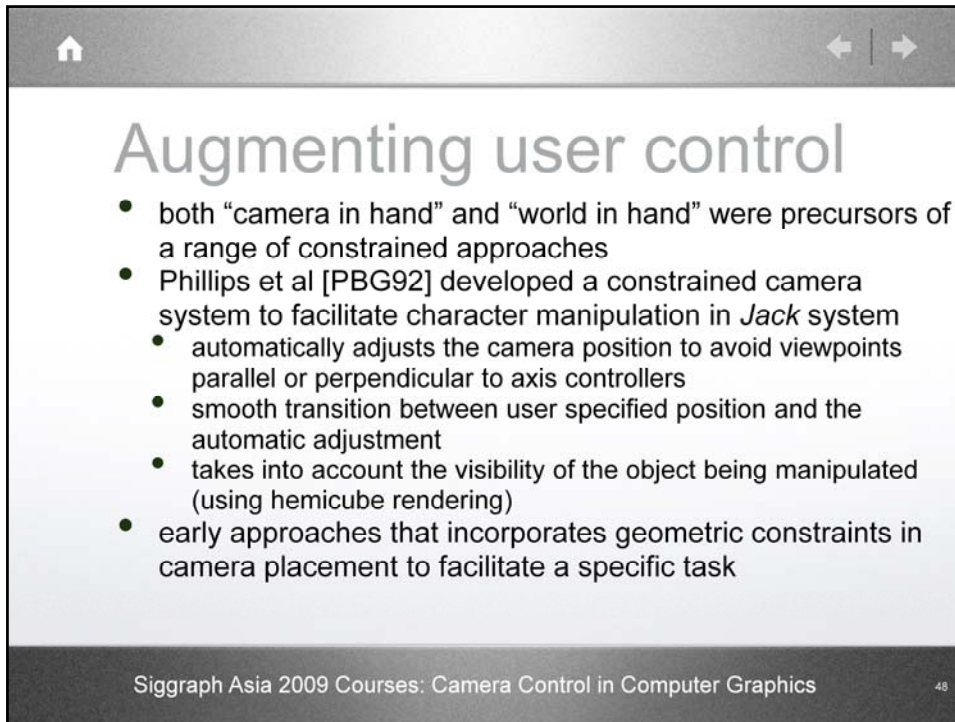
Direct metaphors: high DOF control

- seems likely that an input device with similar degrees of freedom would be the most appropriate input device
- Ware & Osborne [WO90] evaluated a range of 6 dof camera control metaphors utilizing a magnetic tracker:
 - **camera in hand**: the camera is directly manipulated as if it were in the user's hand which encompasses rotational and translational movements
 - **world in hand**: the camera swivels around the world while shooting a fixed point – such as the center of the world (geometrical constraint).
 - **flying vehicle**: the camera can be treated as a the control stick for an airplane, hand motion changes the directional and rotational velocity
- one of the first attempts to evaluate camera control techniques (primarily through subjective reports)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 47

Another metaphor can be added to Ware&Osborne's classification: *walking metaphors* in which the camera moves in the environment while maintaining a constant distance (height) from a ground plane [HW97, FPB87].

Applications tend to use multiple metaphors in sequence (flying vehicle for navigation and world in hand for proximal inspection), and need to propose smooth transitions between them.





Augmenting user control

- both “camera in hand” and “world in hand” were precursors of a range of constrained approaches
- Phillips et al [PBG92] developed a constrained camera system to facilitate character manipulation in *Jack* system
 - automatically adjusts the camera position to avoid viewpoints parallel or perpendicular to axis controllers
 - smooth transition between user specified position and the automatic adjustment
 - takes into account the visibility of the object being manipulated (using hemicube rendering)
- early approaches that incorporates geometric constraints in camera placement to facilitate a specific task

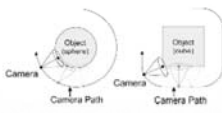
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 48

Techniques have rapidly introduced constraints to augment the usability by assisting the computation of some degrees of freedom. This is typically addressed by reducing the dimensionality of the control problem, and/or the application of physics-based models, vector fields or path planning to constrain possible movement and avoid obstacles [HW97]. For example, the application of a physical model to camera motion control has been explored by Turner et al. [TBGT91]. User inputs are treated as forces acting on a weighted mass (the camera) and friction and inertia are incorporated to damp degrees of freedom that are not the user's primary concern.

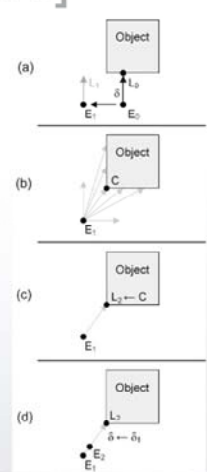


Constraints: Khan [KKS*05]

Khan et al [KKS*05] developed a “hovercam” metaphor for individual **object inspection**:



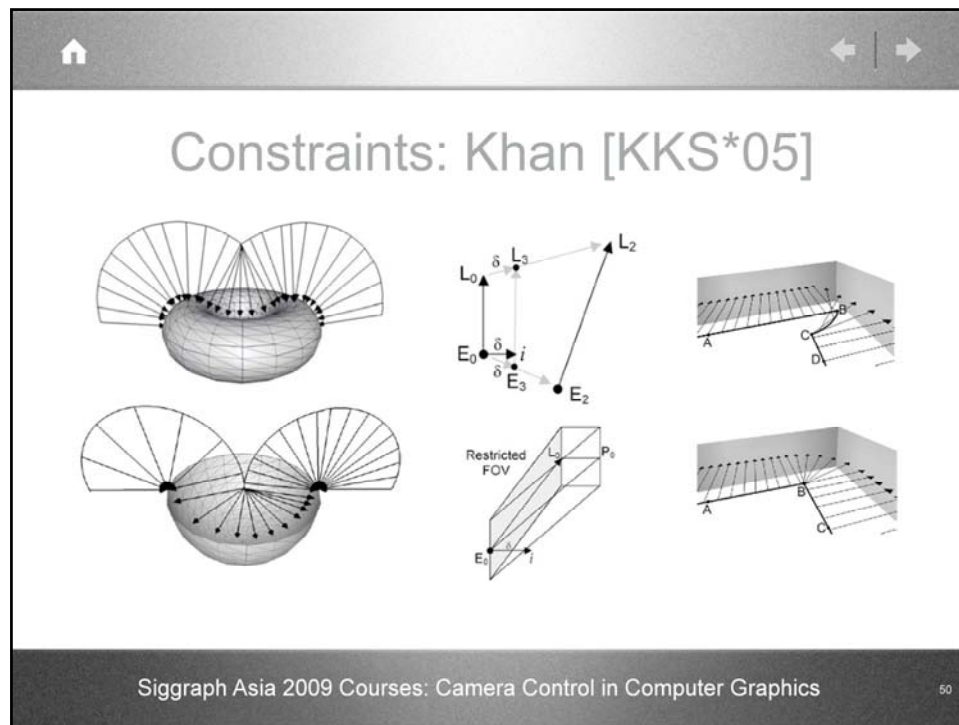
- apply user input to the eye point E_0 (current camera position) and look-at point L_0 , to create E_1 and L_1 ;
- search for the closest point C on the object from the new eye position E_1 ;
- turn the camera to look at C , and
- correct the distance δ_1 to the object to match the original distance to the object δ to generate the final eye position E_2
- clip the distance traveled





Siggraph Asia 2009 Courses: Camera Control in Computer Graphics49

This slide illustrates interactive approaches related to object (referred as proximal inspection) and environment exploration. A certain knowledge of the environment is utilized to assist the user in his navigation or exploration task. Such approaches are split according to their local or global awareness of the 3D scene.

Khan et al. [KKS+05] propose an interaction technique for proximal object inspection that automatically avoids collisions with scene objects and local environments. The hovercam tries to maintain the camera at both a fixed distance around the object and (relatively) normal to the surface, following a hovercraft metaphor. Thus the camera easily turns around corners and pans along at surfaces, while avoiding both collisions and occlusions. Specific techniques are devised to manage cavities and sharp turns.

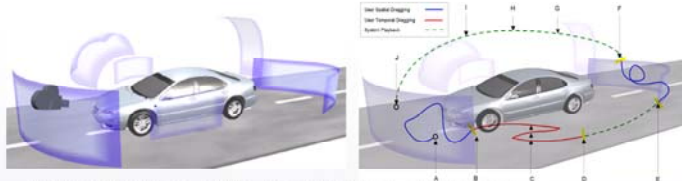


Left, top and bottom: negotiating bumps and holes in proximal inspection
Right, top and bottom: negotiating corners



Constraints: Burtnyk [BKF*02]

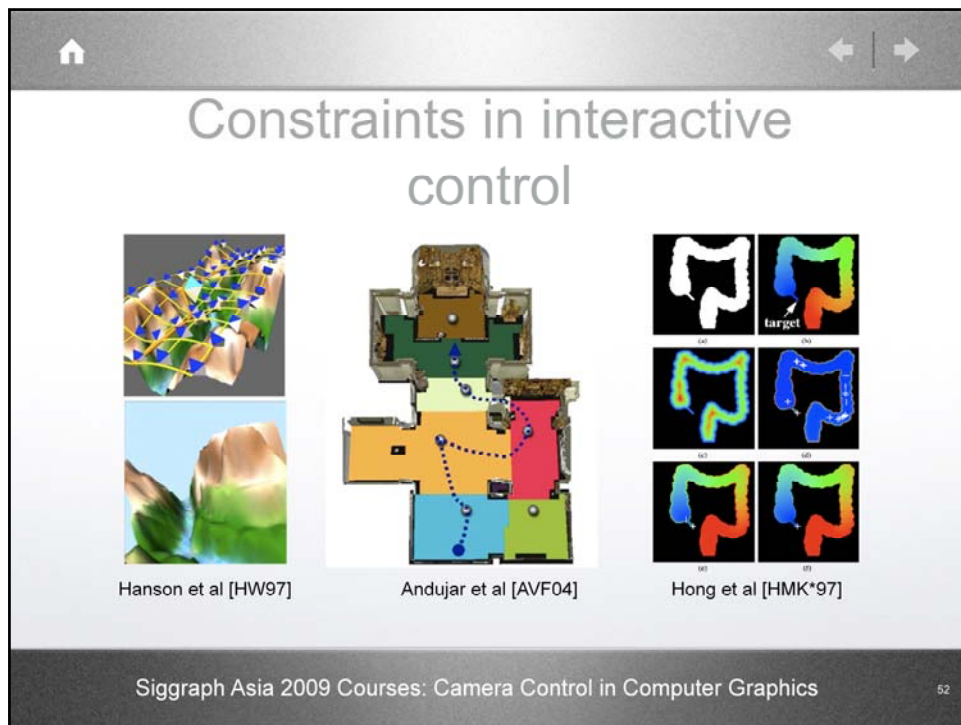
- Burtnyk et al. [BKF*02] developed a system to constrain the camera motions to authored surfaces which are linked with hand-built transitions



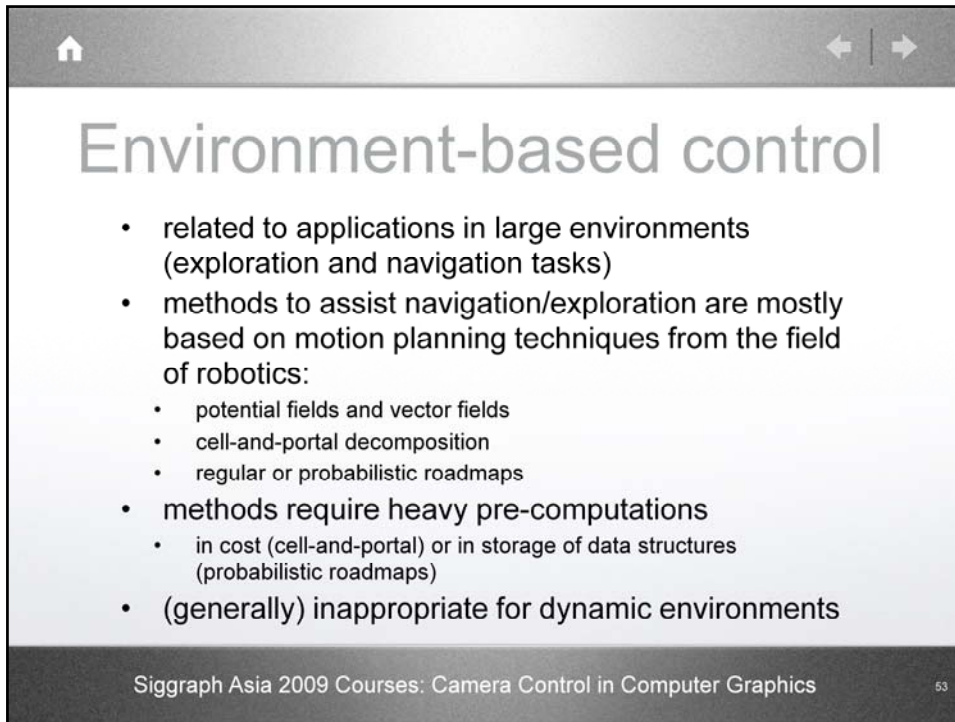
- Interaction as a 2D dragging on surfaces:
 - continuous drag between surfaces and transitions
 - transitions are triggered at specific locations on the constraint surface

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics51

In more stylistic way compared to [KKS*05], Burtnyk et al. [BKF+02] propose an approach in which the camera is constrained to a surface defined around the object to explore (as in [HW97]). The surfaces are designed to constrain the camera to yield interesting viewpoints of the object that will guarantee a certain level of quality in the user's exploratory experience, and automated transitions are constructed between the edges of different surfaces in the scene. The user navigation freely in the bounds of the *constraint surface*, and on reaching an edge is guided to another constraint surface, or hand-built transition.



We now detail techniques that rely on the geometry of the whole environment to build constraints, that assist the users in either navigation or exploration tasks.

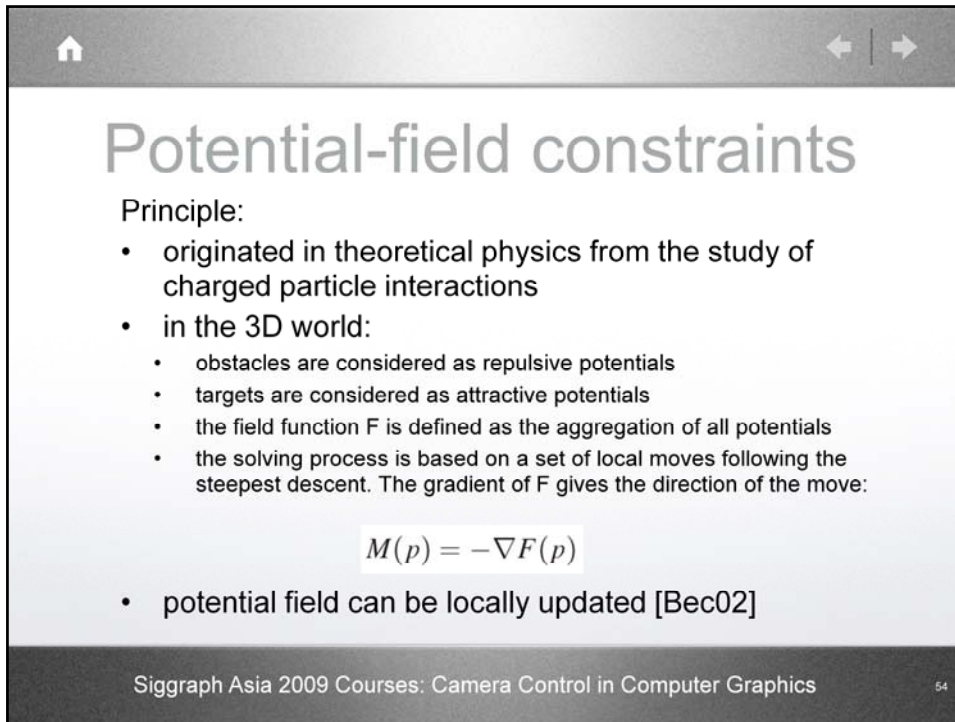


Environment-based control

- related to applications in large environments (exploration and navigation tasks)
- methods to assist navigation/exploration are mostly based on motion planning techniques from the field of robotics:
 - potential fields and vector fields
 - cell-and-portal decomposition
 - regular or probabilistic roadmaps
- methods require heavy pre-computations
 - in cost (cell-and-portal) or in storage of data structures (probabilistic roadmaps)
- (generally) inappropriate for dynamic environments

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 53

Environment-based assistance, for which applications are generally dedicated to the exploration of complex environments, requires specific approaches that are related to the more general problem of path-planning. Applications can be found both in navigation (searching for a precise target) and in exploration (gathering knowledge in the scene). Motion planning problems in computer graphics have mostly been inspired by robotics utilizing techniques such as potential fields, cell decomposition and roadmaps.



The slide is titled "Potential-field constraints" and is part of a presentation, as indicated by the navigation icons at the top. It lists the principles of potential fields and includes a mathematical formula for the move function $M(p)$.

Potential-field constraints

Principle:

- originated in theoretical physics from the study of charged particle interactions
- in the 3D world:
 - obstacles are considered as repulsive potentials
 - targets are considered as attractive potentials
 - the field function F is defined as the aggregation of all potentials
 - the solving process is based on a set of local moves following the steepest descent. The gradient of F gives the direction of the move:



$$M(p) = -\nabla F(p)$$

- potential field can be locally updated [Bec02]

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 54

The low cost of implementation and evaluation of potential fields make them a candidate for applications in real-time contexts.


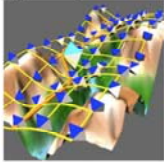
The efficiency of the method is however overshadowed by its limitations with respect to the management of local minima as well as difficulties incorporating highly dynamic environments. Nonetheless, some authors have proposed extensions such as Beckhaus [Bec02] who relies on dynamic potential fields to manage changing environments by discretizing the search space using a uniform rectangular grid and therefore only locally re-computing the potentials.



Vector-field constraints (1)

Extension of potential fields [HW97, XH98]:

- requires a **constraint surface**: navigation surface that establishes the reachable areas in the environment
- requires a **camera model field**: to each point of the constraint surface is attached some orientation constraint
- a bi-cubic spline interpolation is used to interpolate between the user specified orientation constraints
- possibility to automate the process [TC01]



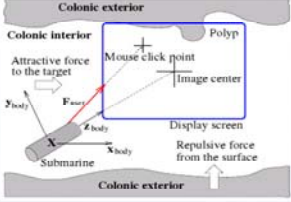
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 55

In [HW97], the constraint surface is defined by the user, together with a number of orientation key-points. Recent approaches consider automated computation of either scalar or vector fields to assist the users both in location and orientation [TC01, ETT07]. This requires to answer a number of key issues (handling bottlenecks such as narrow doorways, handling large open spaces, identifying essential landmarks that make this problem a difficult one).

⌂
◀ | ▶

Vector-field constraints (2)

- Application to virtual colonoscopy [HMK97]:
 - repulsive forces to maintain the camera away from the colonic surfaces: $V(\mathbf{X})$
 - attractive forces towards the target: $V(\mathbf{X})$
 - energy dissipation factor: k_l
 - rigid body dynamics to handle user interaction: \mathbf{F}_{user}





$$\dot{\mathbf{P}}(t) = -\nabla V(\mathbf{X}) - k_l \mathbf{P}(t) + \mathbf{F}_{user}(t),$$

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics
56

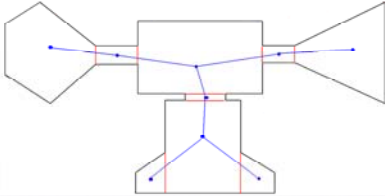
Virtual endoscopy enables the exploration of the internal structures of a patient's anatomy. Difficulties arise in the interactive control of the camera within the complex internal structures. Ideally important anatomical features should be emphasized and significant occlusions and confined spaces avoided. The underlying techniques mostly rely on skeletonization of the structures and on path planning approaches such as potential fields. For example, [HMK97] and [CHL+98] report a technique that avoids collisions for guided navigation in the human colon. The surfaces of the colon and the center line of the colon are modeled with repulsive and attractive fields respectively.

In [HMK97], the camera is guided by some repulsive forces from the colonic surface, attractive ones that push the camera towards a given target, and user inputs (when pointing an area on the surface). The process is however very specific to the problem (a more general geometry would lead to many cases of failure or inappropriate guidance).



Cell-and-portal decomposition

- performs partitions of the environment into sub-regions (the **cells**), and connections between sub-regions (the **portals**)





- an adjacency graph is built by connecting cells
- camera exploration/navigation tasks can then be casted as a planning process in the adjacency graph [AVF04]

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

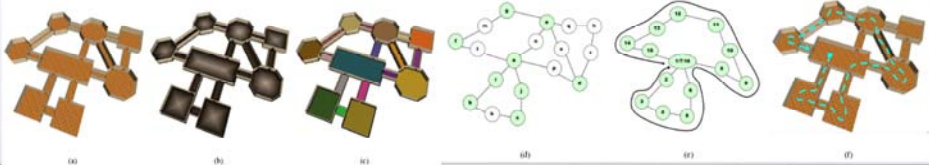
57

Cell decomposition approaches split the environment into spatial regions (cells) and build a network that connects the regions. Navigation and exploration tasks utilize this cell connectivity while enforcing other properties on the camera. For example, [AVF04] proposed such a technique to ease the navigation process and achieve shots of important entities and locations. Using a cell-and portal decomposition of the scene together with an entropy-based measure of the relevance of each cell critical way-points for the path could be identified.





Cell-and-portal decomposition

- provides a *structure* to the environment, to better perform navigation/walkthrough tasks (the decomposition can be guided by the user for specific needs)
- Andujar et al. [AVF04] employ this structure to:
 - identify the individual interest of each cell (with an entropy-based metric)
 - compute the sequence of most relevant cells to visit
 - compute a path connecting the cells, portals and relevant viewpoints in the cells



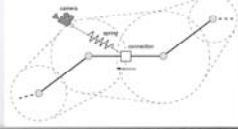


(a) (b) (c) (d) (e) (f)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 58



Voxel-based decomposition



- a regular partitioning of the free space (voxels) can be used to generate guided tours [ETT07]:
 - visibility of (authored) landmarks is computed for each of the voxels in a pre-process
 - all voxels that view at least one landmark are connected together to form an adjacency graph
 - a solving process (Travel Salesman-like) computes the suite of voxels to visit in the graph to ensure that each landmark has been viewed at least once
 - in interactive mode, a memory of the visited landmarks is maintained to guide/constrain the users navigation, through a spring-based physical system



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

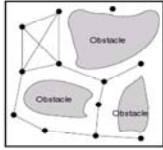
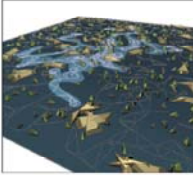
59

Following an idea similar to Andujar, yet in a more interactive context, Elmquist et al. [ETT07] propose to automate the construction of a navigation graph between user-defined landmarks. The environment is decomposed into voxels, each of which is evaluated for visibility against the landmarks. An adjacency graph is then built between voxels sharing the same landmarks, and explored with a TSP algorithm to compute the best path that visits all the landmarks.



Roadmap constraints (1)

- roadmap planners operate in two phases:
 - first sampling the space of possible configurations
 - second constructing a connectivity graph by linking neighbour samples (and checking for collision on the links)





- simple to construct and navigate inside the graph
- complex to determine the appropriate density of sampling (but PRM complexity is a factor of the scene complexity)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

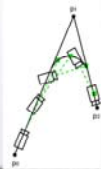
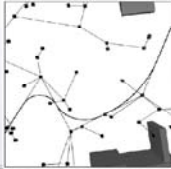
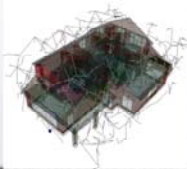
60

Roadmaps, and especially probabilistic roadmaps are a simple-to-implement and efficient technique to perform path planning tasks at the level of an environment. For transition planning (moving from one landmark to another), target tracking and cut-jumping (switching between viewpoints), the process needs to be augmented by visibility computation, either in a static way [NO03], or in a dynamic way [LC08].





Roadmap constraints (2)

- [NO03] rely on probabilistic roadmap techniques for camera planning:
 - roadmap is consisting of collision-free camera motions (the camera is abstracted as a sphere, the motion as a cylinder)
 - planning is performed with an advanced Dijkstra process (avoids sharps turns)
 - path is smoothed and camera orientation anticipates camera motion



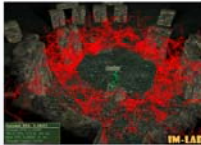
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 61

In [NO03] visibility is guaranteed between connected nodes. Such PRMs can be used in an interactive approach by selecting the most appropriate given the current configuration and the user inputs. The main drawback lies in the cost of updating the data structure when considering dynamic elements.



A roadmap-based approach

- Using a locally defined PRM [LC08]
 - a probabilistic roadmap is computed around the target (and moves with the target)
 - path planning is performed with the probabilistic roadmap (lazy evaluation of collision and occlusion):
 - collision/occluded nodes are removed
 - new nodes are inserted with a density parameter
- provides a more robust approach (global idea of the visibility, cuts between viewpoints)

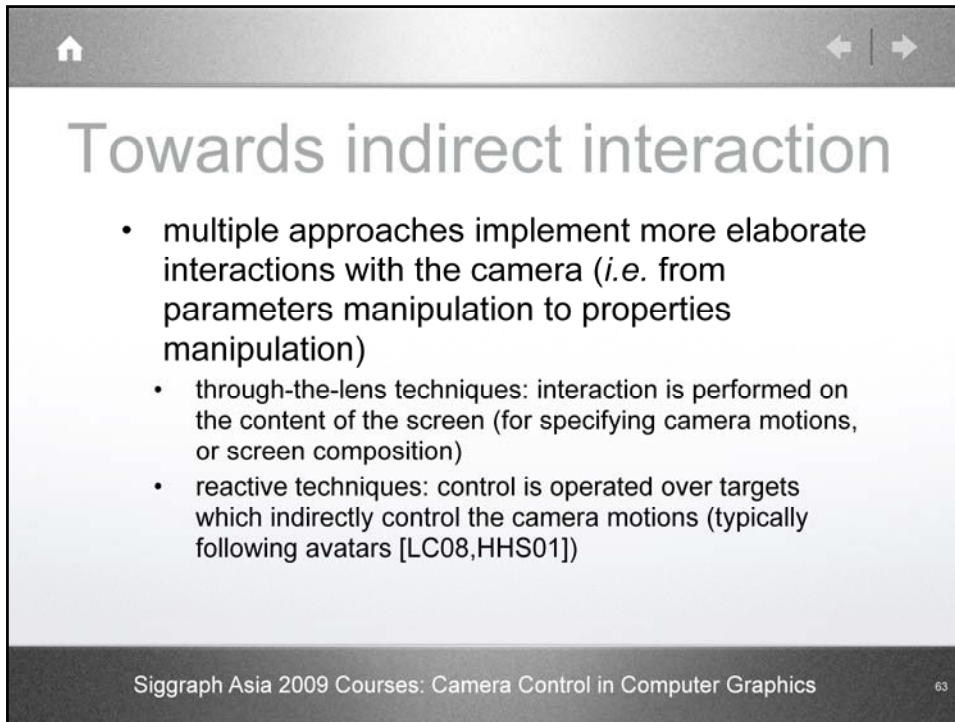


Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 62

Previous approaches generally suffer from their locality (searching for viewpoints in the local neighborhood of the current camera location). Chang and Li introduce a probabilistic roadmap technique that helps to reduce this locality:

- a roadmap is defined in the local basis of the camera target (the roadmap is built once, and then is only locally modified)
- paths are searched for in this roadmap by evaluating every configuration wrt. visibility of the target and possible collision of the path with the environment:
 - occluded viewpoints and non reachable viewpoints are removed from the roadmap
 - new viewpoints are added when necessary
- in critical situations, cuts can be performed between viewpoints (cuts are represented as expensive edges in the roadmap)

Provides a reactive approach that is more global (lazy-evaluation of the knowledge in connected edges), and allows cuts.



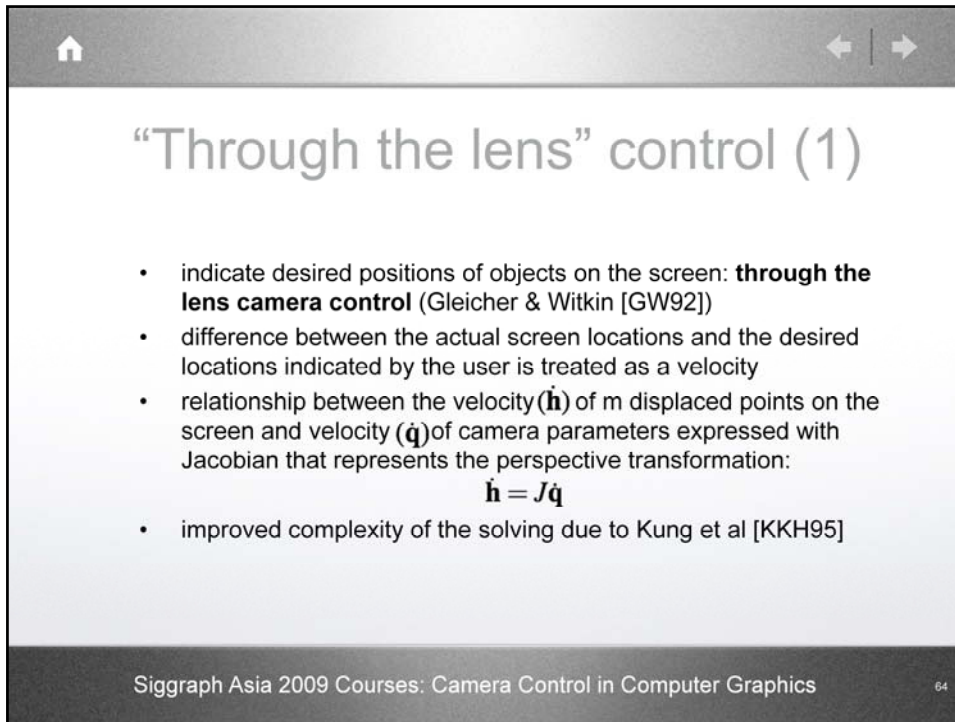
Towards indirect interaction

- multiple approaches implement more elaborate interactions with the camera (*i.e.* from parameters manipulation to properties manipulation)
 - through-the-lens techniques: interaction is performed on the content of the screen (for specifying camera motions, or screen composition)
 - reactive techniques: control is operated over targets which indirectly control the camera motions (typically following avatars [LC08,HHS01])

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

63

In moving further away from the direct manipulation of camera parameters, through-the-lens techniques enable the control of the screen content.

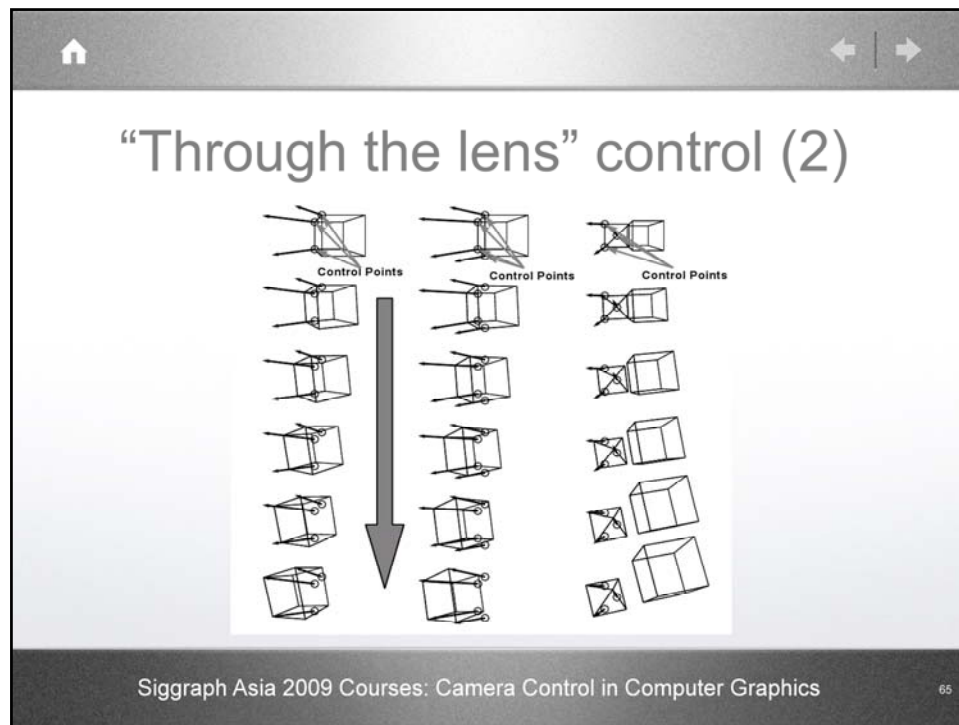





The slide is titled "Through the lens" control (1). It features a list of four bullet points. The first bullet point mentions "through the lens camera control" and cites "Gleicher & Witkin [GW92]". The second bullet point discusses the difference between actual and desired screen locations being treated as a velocity. The third bullet point describes the relationship between the velocity of points on the screen and camera parameters, expressed with a Jacobian, and includes the equation $\dot{\mathbf{h}} = J\dot{\mathbf{q}}$. The fourth bullet point mentions the improved complexity of solving due to Kung et al [KKH95]. The footer of the slide contains the text "Siggraph Asia 2009 Courses: Camera Control in Computer Graphics" and the number "64".

“Through the lens” control (1)

- indicate desired positions of objects on the screen: **through the lens camera control** (Gleicher & Witkin [GW92])
- difference between the actual screen locations and the desired locations indicated by the user is treated as a velocity
- relationship between the velocity ($\dot{\mathbf{h}}$) of m displaced points on the screen and velocity ($\dot{\mathbf{q}}$) of camera parameters expressed with Jacobian that represents the perspective transformation:
$$\dot{\mathbf{h}} = J\dot{\mathbf{q}}$$
- improved complexity of the solving due to Kung et al [KKH95]

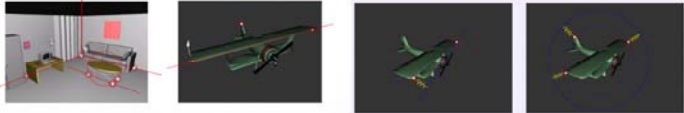
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 64





“Through the lens” control (3)



- multiple primitives (lines, circles, paths) can be used to control the composition [YCL08]



- a mass-spring interaction metaphor dampers the users manipulations (often over-constrained)
- composition can be further manipulated by interacting with light sources, shadows and penumbras together with camera parameters

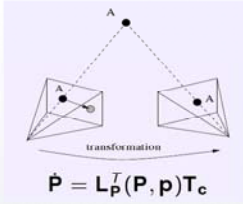
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

66



Visual servoing techniques (1)

- use robotics techniques to control camera motions through screen-space constraints [MC02]
- the relation between the changes in 3D environment and the changes in the 2D projected image is expressed in the image Jacobian


$$\dot{\mathbf{P}} = \mathbf{L}_P^T(\mathbf{P}, \mathbf{p}) \mathbf{T}_c$$

- key idea is to invert the equation (constraining the velocity of the key features, compute the camera velocity)

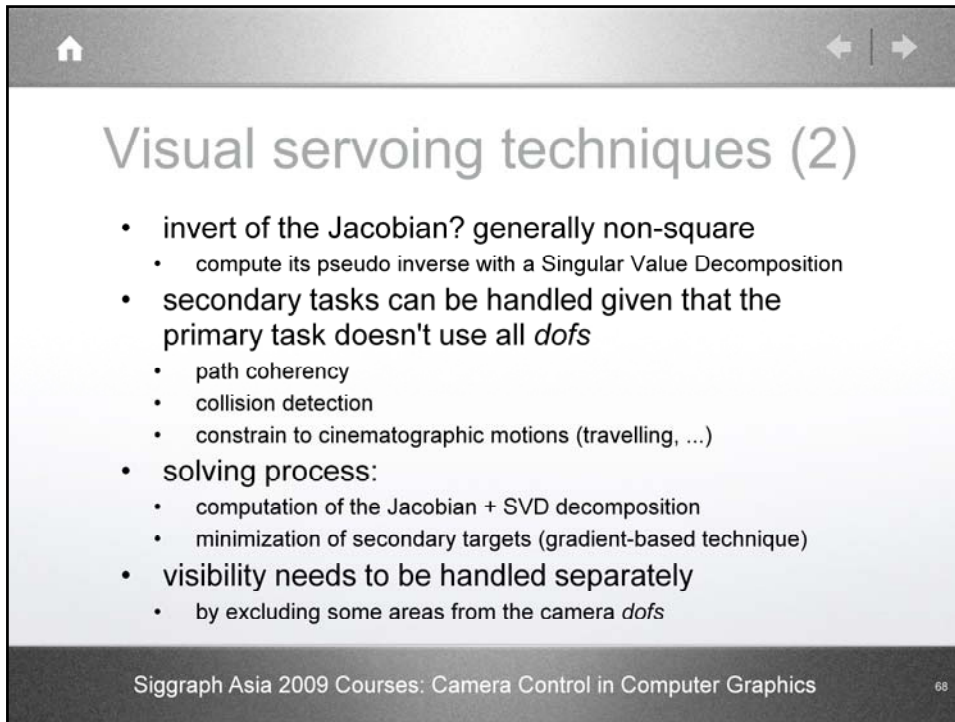
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

67

Visual servoing techniques relies on the regulation in the final image of a set of visual features (points, segments, lines).

The image Jacobian (L) expresses the link between the motion of a visual features (P) in the 2D screen and the motion of the camera (it's a linearization of the projection relation for the camera configuration).

The key idea is then to invert the equation, in order to express the variation on camera parameters that correspond to a desired motion of the visual feature on the screen. For exemple, in order to constrain a mobile 3D point at a given location on screen, requires to solve $Jq=0$ at every frame.



Visual servoing techniques (2)

- invert of the Jacobian? generally non-square
 - compute its pseudo inverse with a Singular Value Decomposition
- secondary tasks can be handled given that the primary task doesn't use all *dofs*
 - path coherency
 - collision detection
 - constrain to cinematographic motions (travelling, ...)
- solving process:
 - computation of the Jacobian + SVD decomposition
 - minimization of secondary targets (gradient-based technique)
- visibility needs to be handled separately
 - by excluding some areas from the camera *dofs*

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 68

The Jacobian matrix is generally non square ($m \times n$):

- m is the number *dofs* of the camera (7 for euler-based, 8 for quaternion-based)

- n is the number of parameters of the visual features in 2D (2 for a point, 3 for a line, 4 for a segment)



The pseudo inverse of the matrix can be computed by Singular Value Decomposition which is in $O(mn^2)$.

If all camera *dofs* are not constrained, one can perform secondary tasks (see details in next slide) through a minimization process.

Solving process is quite efficient (cost of Jacobian + SVD + minimization).

However:

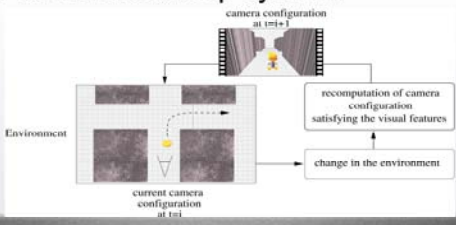
- difficult to balance between primary and secondary tasks
- some tasks cannot be easily expressed as a minimization process (visibility/occlusion)



Reactive techniques

Reactive = indirect approach to camera control (the target is controlled, the camera reacts)

- mainly devoted to target tracking
 - primary concern is visibility (not losing the target)
 - secondary concerns are continuity and frame coherency
- modelled as a closed-loop system:



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

69

A technique of choice in real-time gaming environments with 3rd person view

The camera is linked to the character with a virtual rod (more critical with multiple characters), which reduces the problem to 3 degrees of freedom in a spherical coordinate system (radius to target, elevation, azimuth)

In complex environments the first main concern is to maintain – in a continuous and coherent way -- the visibility of the target (see Part V), and the second is related to the playability (how to adapt the user controls when the viewpoint changes).

In games, critical issues do appear in contrived configurations :

- camera blocked in a corner as the character backs up
- camera fails to maintain viewpoint on target of interest (eg, shooting an opponent)

These issues are linked to the (restricted) local knowledge of the environment (in terms of visibility and accessibility).

Shares a set of common issues with the domain of robotics (planning, visibility, prediction)

🏠
◀ | ▶

Incremental solving approach

- Based on algebraic computations [HHS01]
 - a closed set of constraints is considered: camera height, angle of interest, size, visibility, screen position
 - solving is based on the incremental satisfaction of each constraint:

```

graph LR
    A[height angle] --> B[facing]
    C[level at] --> D[angle of interest]
    B --> E[size]
    D --> E
    E --> F[visibility]
    F --> G[lookat]
            
```

Ordered algebraic steps to compute a camera configuration [Halper et al. 01]

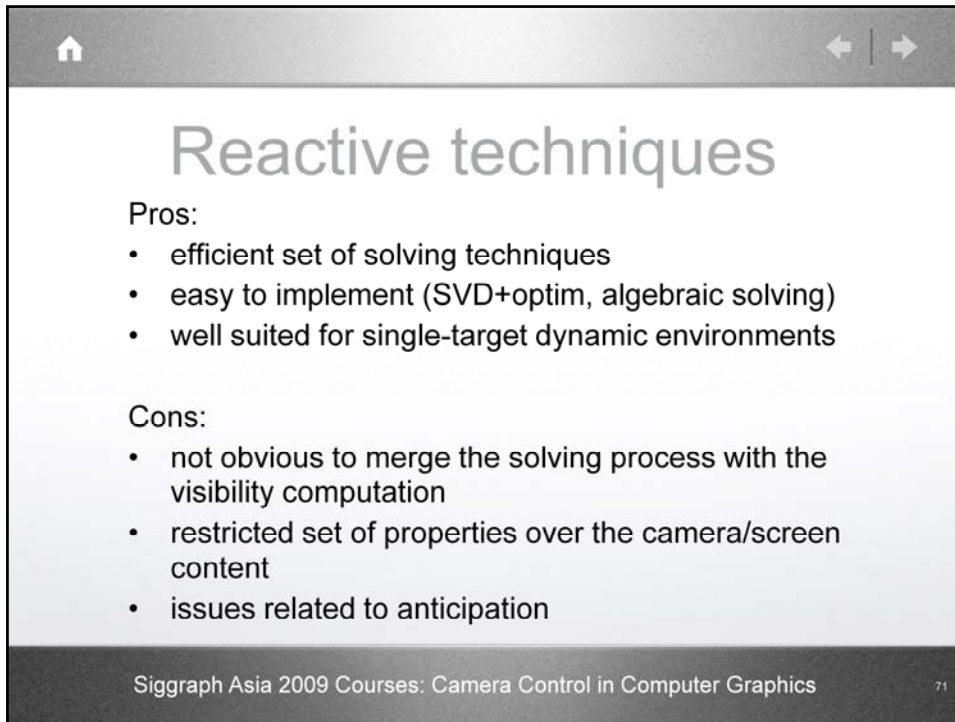
- particular effort is expended on enforcing coherency
- occlusion is computed by hardware rendering Potential Visibility Sets (see Visibility section)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics
70

Halper et al. propose an incremental solving technique for reactive camera control. A small set of constraints is proposed, and each constraint is solved with an algebraic expression (except visibility – see Part V). Constraints are solved in an incremental way (each constraint is applied on the result of the previous constraint), based for example on a spherical coordinate system around the target object:

- camera height: the camera is located at a specified height, relative to the target (fixes **elevation**)
- angle of interest: the camera is rotated around the target to satisfy the requirement (fixes **azimuth**)
- size: constrains the distance to the target (fixes **radius**)
- visibility: moves the camera to a location that maximises visibility (see Part V)
- Look at: constrains the orientation of the camera (ie camera quaternion).

The approach is efficient (simple algebraic computations, visibility is computed with hardware rendering in low resolution buffers). However the technique is not that easy to extend to multiple targets, and can suffer from the local knowledge it has of its environment.



Reactive techniques

Pros:

- efficient set of solving techniques
- easy to implement (SVD+optim, algebraic solving)
- well suited for single-target dynamic environments

Cons:

- not obvious to merge the solving process with the visibility computation
- restricted set of properties over the camera/screen content
- issues related to anticipation

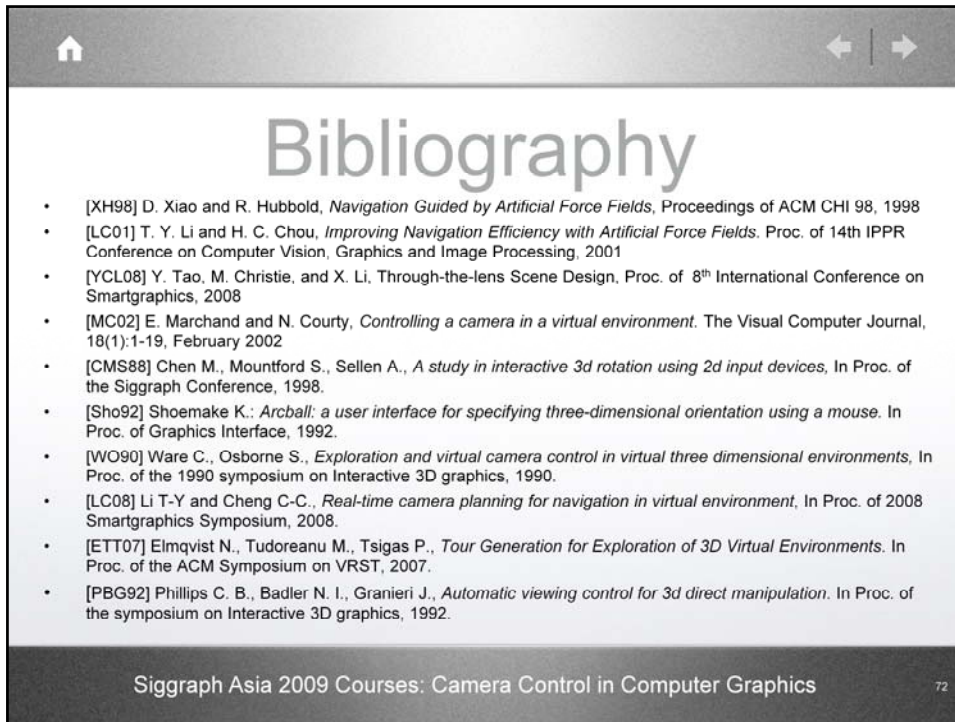
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 71

Reactive approaches represent a set of techniques of first choice in gaming environments: easy to compute and relatively inexpensive. Main issues are related to:

- how to properly handle visibility – and how this step can be incorporated into the solving pipeline
- how to handle frame/path coherency. No metrics are provided. Systems generally rely on a physical model (mass spring) to damper the camera motions under sudden changes. Coherency is strongly related to the need of anticipating the motions of the targets with relation to the environment. Very few anticipation models are provided (and by nature are very specific).

Screen coherency could be taken into account by provided some measure of the variation/acceleration of screen velocities. Screen coherency could actually be decomposed into :

- predictive power (how well does the camera perform in anticipating occlusion)
- stability (how stable is the camera in critical situations)
- responsiveness (how fast does the camera handle sudden changes)

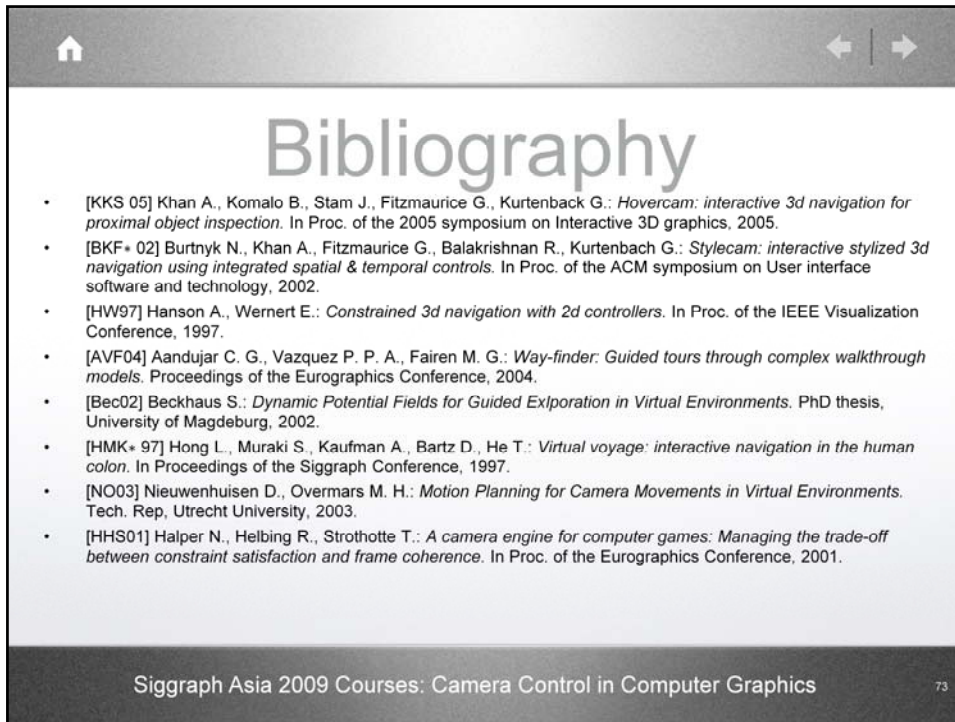


The slide features a header bar with a home icon on the left and navigation arrows on the right. The main title 'Bibliography' is centered in a large, light gray font. Below the title is a list of 12 references, each preceded by a bullet point. The references are formatted with author initials in brackets, followed by the author names and the publication details. The slide has a dark gray footer bar containing the course title and the page number 72.

Bibliography

- [XH98] D. Xiao and R. Hubbard, *Navigation Guided by Artificial Force Fields*, Proceedings of ACM CHI 98, 1998
- [LC01] T. Y. Li and H. C. Chou, *Improving Navigation Efficiency with Artificial Force Fields*, Proc. of 14th IPPR Conference on Computer Vision, Graphics and Image Processing, 2001
- [YCL08] Y. Tao, M. Christie, and X. Li, *Through-the-lens Scene Design*, Proc. of 8th International Conference on Smartgraphics, 2008
- [MC02] E. Marchand and N. Courty, *Controlling a camera in a virtual environment*, The Visual Computer Journal, 18(1):1-19, February 2002
- [CMS88] Chen M., Mountford S., Sellen A., *A study in interactive 3d rotation using 2d input devices*, In Proc. of the Siggraph Conference, 1998.
- [Sho92] Shoemake K.: *Arcball: a user interface for specifying three-dimensional orientation using a mouse*. In Proc. of Graphics Interface, 1992.
- [WO90] Ware C., Osborne S., *Exploration and virtual camera control in virtual three dimensional environments*, In Proc. of the 1990 symposium on Interactive 3D graphics, 1990.
- [LC08] Li T-Y and Cheng C-C., *Real-time camera planning for navigation in virtual environment*, In Proc. of 2008 Smartgraphics Symposium, 2008.
- [ETT07] Elmqvist N., Tudoreanu M., Tsigas P., *Tour Generation for Exploration of 3D Virtual Environments*. In Proc. of the ACM Symposium on VRST, 2007.
- [PBG92] Phillips C. B., Badler N. I., Granieri J., *Automatic viewing control for 3d direct manipulation*. In Proc. of the symposium on Interactive 3D graphics, 1992.

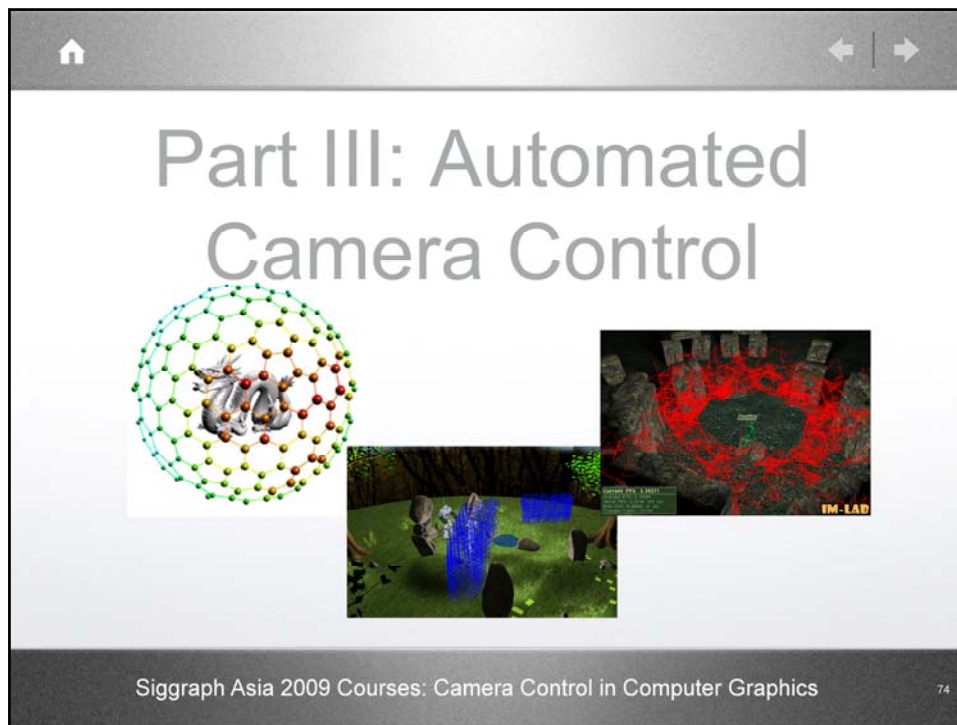
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 72





Bibliography

- [KKS 05] Khan A., Komalo B., Stam J., Fitzmaurice G., Kurtenbach G.: *Hovercam: interactive 3d navigation for proximal object inspection*. In Proc. of the 2005 symposium on Interactive 3D graphics, 2005.
- [BKF* 02] Burtnyk N., Khan A., Fitzmaurice G., Balakrishnan R., Kurtenbach G.: *Stylecam: interactive stylized 3d navigation using integrated spatial & temporal controls*. In Proc. of the ACM symposium on User interface software and technology, 2002.
- [HW97] Hanson A., Wernert E.: *Constrained 3d navigation with 2d controllers*. In Proc. of the IEEE Visualization Conference, 1997.
- [AVF04] Aandujar C. G., Vazquez P. P. A., Fairen M. G.: *Way-finder: Guided tours through complex walkthrough models*. Proceedings of the Eurographics Conference, 2004.
- [Bec02] Beckhaus S.: *Dynamic Potential Fields for Guided Exploration in Virtual Environments*. PhD thesis, University of Magdeburg, 2002.
- [HMK* 97] Hong L., Muraki S., Kaufman A., Bartz D., He T.: *Virtual voyage: interactive navigation in the human colon*. In Proceedings of the Siggraph Conference, 1997.
- [NO03] Nieuwenhuisen D., Overmars M. H.: *Motion Planning for Camera Movements in Virtual Environments*. Tech. Rep. Utrecht University, 2003.
- [HHS01] Halper N., Helbing R., Strothotte T.: *A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence*. In Proc. of the Eurographics Conference, 2001.

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 73

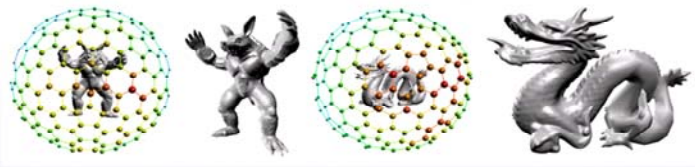


This section will provide an overview of applications and techniques related to the automated control of camera parameters, i.e. not considering any user interaction in the process, should it be direct or not. Such approaches mainly refer to topics such as viewpoint computation (models and techniques to estimate the amount of information a viewpoint contains), declarative approaches (where a description of the expected result is provided and solvers generate solution sets), and automated path computations (which consists in reasoning from the information in the scene, possibly enriched by narrative elements, to generate appropriate shots, paths and edits).



Viewpoint computation (1)

- how much information of a scene is in the viewpoint:
 - highly subjective and domain dependent
 - notion of viewpoint entropy [VFSH01]



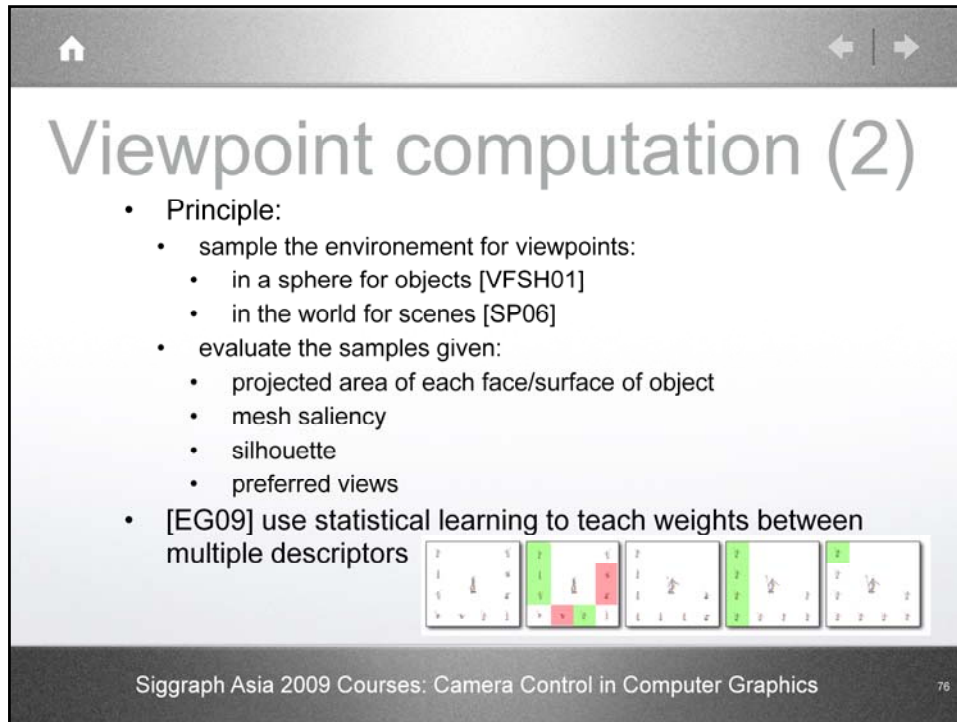
- based on the projected area of faces/objects
 - tessellation independent views [VA07]
 - viewpoint saliency [LVJ05]

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

75

Viewpoint computation finds many applications in computational geometry, visual servoing, active vision and image-based rendering (IBR). In IBR, the technique can be used to generate multiple impostors for an object (while enforcing minimal set of views, and maximum covering of the object);

Mesh saliency was introduced as a curvature-based metric. The authors proposed a gradient-descent optimization to compute views that maximise this saliency.



The slide is titled "Viewpoint computation (2)" and contains a list of principles for viewpoint computation. The list is as follows:

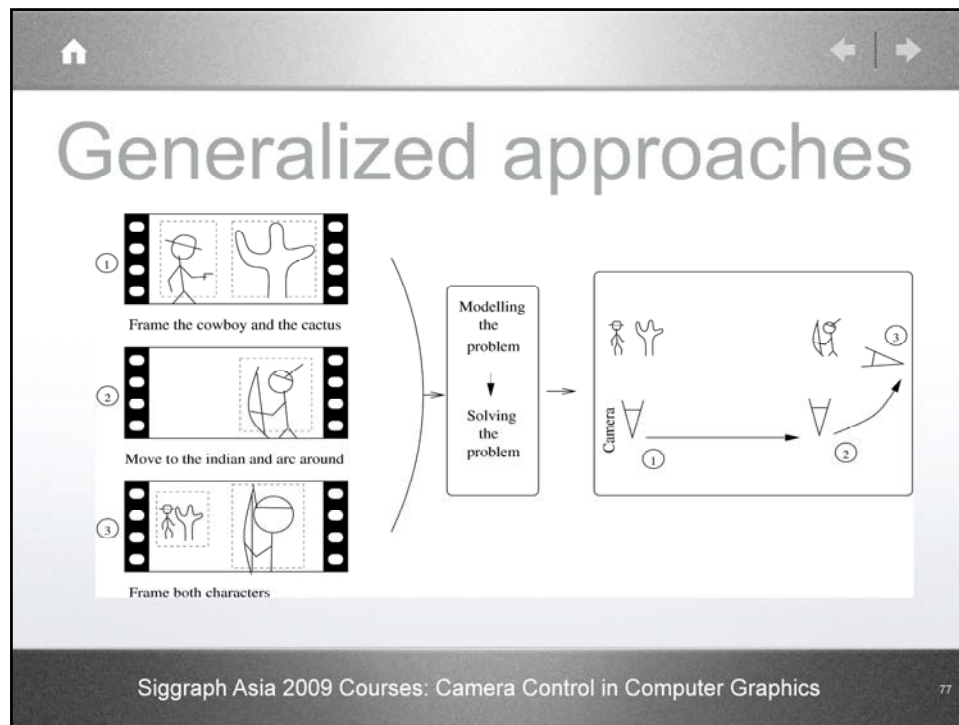
- Principle:
 - sample the environment for viewpoints:
 - in a sphere for objects [VFSH01]
 - in the world for scenes [SP06]
 - evaluate the samples given:
 - projected area of each face/surface of object
 - mesh saliency
 - silhouette
 - preferred views
 - [EG09] use statistical learning to teach weights between multiple descriptors

Below the list, there is a small diagram consisting of five square panels. Each panel shows a 3D object (a teapot) in a different orientation. The panels are labeled with numbers 1 through 5. The first panel shows the teapot from a side view. The second panel shows the teapot from a top-down view. The third panel shows the teapot from a front view. The fourth panel shows the teapot from a back view. The fifth panel shows the teapot from a side view, rotated 90 degrees. The panels are arranged in a row, and the numbers 1 through 5 are written below each panel.

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 76

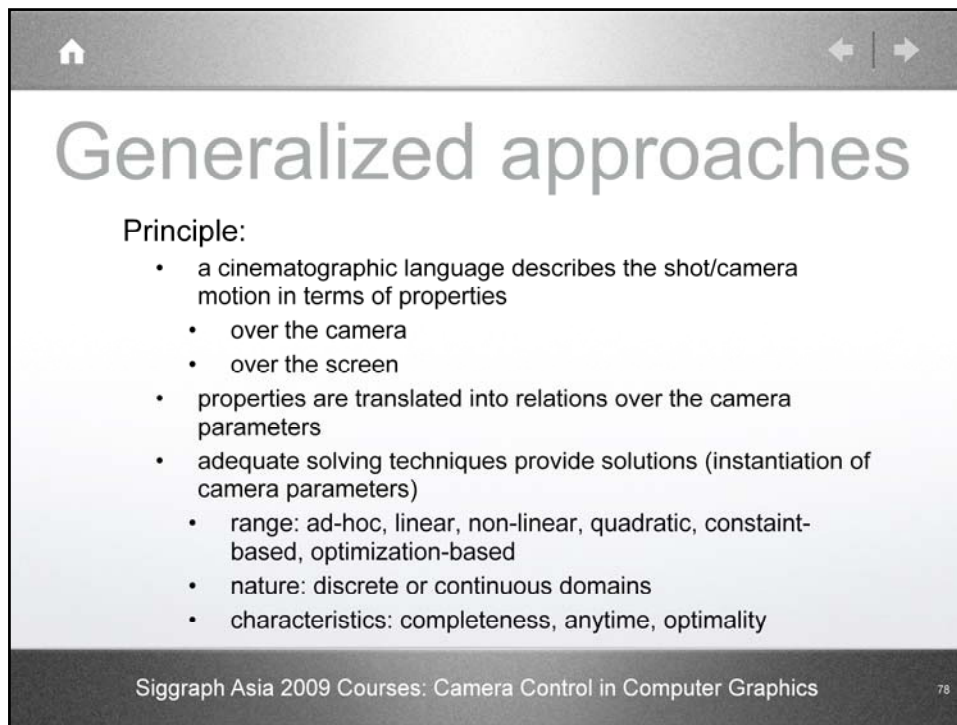
Criteria to compute good views are obviously numerous/ Literature generally consider:

- silhouette (good metric in perception for familiar objects)
- mesh saliency (saliency is essential in the preception process -- recognizability)
- surface area
- importance of preferred views (related to the fonctionnality of the object)
- composition constraints



Generalized approaches would not restrict to only character tracking or user interaction, but essentially attempt to cover, in computer graphics, the full range of applications a camera covers in real-life. Generalized approaches are based on the idea that shots can be expressed in terms of high-level (specific or general) properties, which in turn are expressed as constraints or optimization functions over the camera degrees of freedom.

This is a clear move towards more expressiveness, and a more cinematographic experience in computer graphics, by considering a large set of properties, by looking at composition and narrative aspects



The slide is titled "Generalized approaches" in a large, bold, sans-serif font. Below the title, the word "Principle:" is followed by a bulleted list. The list contains six main items, with the last one having three sub-bullets. The slide has a dark grey header with a home icon and navigation arrows, and a dark grey footer with the course name and the number 78.

Generalized approaches

Principle:

- a cinematographic language describes the shot/camera motion in terms of properties
 - over the camera
 - over the screen
- properties are translated into relations over the camera parameters
- adequate solving techniques provide solutions (instantiation of camera parameters)
 - range: ad-hoc, linear, non-linear, quadratic, constraint-based, optimization-based
 - nature: discrete or continuous domains
 - characteristics: completeness, anytime, optimality

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 78



A shot is described in terms of properties:

- over the camera (high angle, low angle, travelling, panoramic ...)
- over the content of the screen (exact and relative locations of objects, orientations of objects, visibility, size and depths)

Language is generally adapted from cinematography/television and photography literature.


Properties are then written in terms of constraints or fitness functions defined over the set of camera parameters (or path parameters). A range of techniques exist, from simple enumerate/evaluate, to interval-based constraint techniques, optimization and constrained-optimization.

The choice of the solving techniques mainly relies on the application its dedicated to.




Step1: Describe

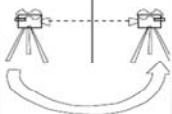
- Properties over the camera motion
 - travelling (tracking, dolly)
 - panoramic
 - arcing
 - zoom
- Properties over the camera parameters
 - vantage angle (high, low)
 - height
 - aperture / roll
- Properties over the content of the screen
 - framing / visibility
 - depth
 - relative location




Panoramic



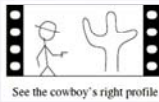
Travelling (tracking ou dolly)



Arcing



View the two characters



See the cowboy's right profile

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

79

The description can be directly inspired from cinematography and photography literature, or may correspond to any need specific to the devised application.

Motions:

- arcing represents a rotational motion around a target (or set of targets)
- travelling is a linear motion (tracking when the camera dof is parallel to the motion, and dolly when its orthogonal)

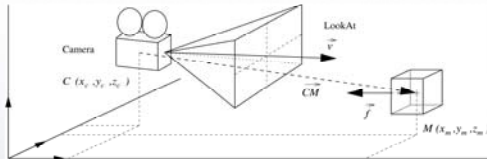
Content of the screen:

- framing is commonly represented by a rectangular shape that bounds the location of the character/target – other primitives can be used (see CamPlan).
- visibility is not trivial to characterize (partial occlusion eg. through leaves or fences, or short-term occlusion)

🏠
⬅️ | ➡️

Step2: Model

- How to express high-level properties into constraints/fitness functions?
 - which underlying representation for the objects?
 - which level of stiffness in expressing the properties?
- Example 1: the orientation property



$$\mathbf{q} = [x, y, z, \phi, \theta, \psi, \gamma]^T$$

$$f_1(\mathbf{q}) = 1 - (v \cdot f + 1)/2$$

- Expressed as a dot product between the camera \mathbf{q} and the intrinsic orientation of the target

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics
80

Main questions when modelling the problem are:

- which representations (or abstractions) should be chosen for the objects? eg. what would be the intrinsic orientation of a character: body or face? which bounding primitives should be used?
- which level of stiffness? When translating properties into constraints, some degree of variation should be allowed (otherwise problems become rapidly over-constrained). For the orientation property, for example, is it required to be exactly on the front axis of the target object, or can any value (in a range, or in a range with a preferred value) be used? Then are there some overlaps with other orientation properties (eg ¾ left profile).?

🏠
⏪ | ⏩

Step 2: Model

- Example 2: the framing property

- constraining point s(xs,ys,zs) to project into area [x1,x2,y1,y2]

with $[x', y']^T = H(\mathbf{q}) \cdot [x_P, y_P, z_P]^T = P(\gamma) \cdot R([\phi, \theta, \psi]^T) \cdot T([x, y, z]^T) \cdot [x_P, y_P, z_P]^T$

$$\begin{cases} c_1 : x' \geq x_1 \\ c_2 : x' \leq x_2 \\ c_3 : y' \geq y_1 \\ c_4 : y' \leq y_2 \end{cases}$$

- $H(\mathbf{q})$ is strongly non-linear
- how to handle the projection of a complex shape?

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics
81

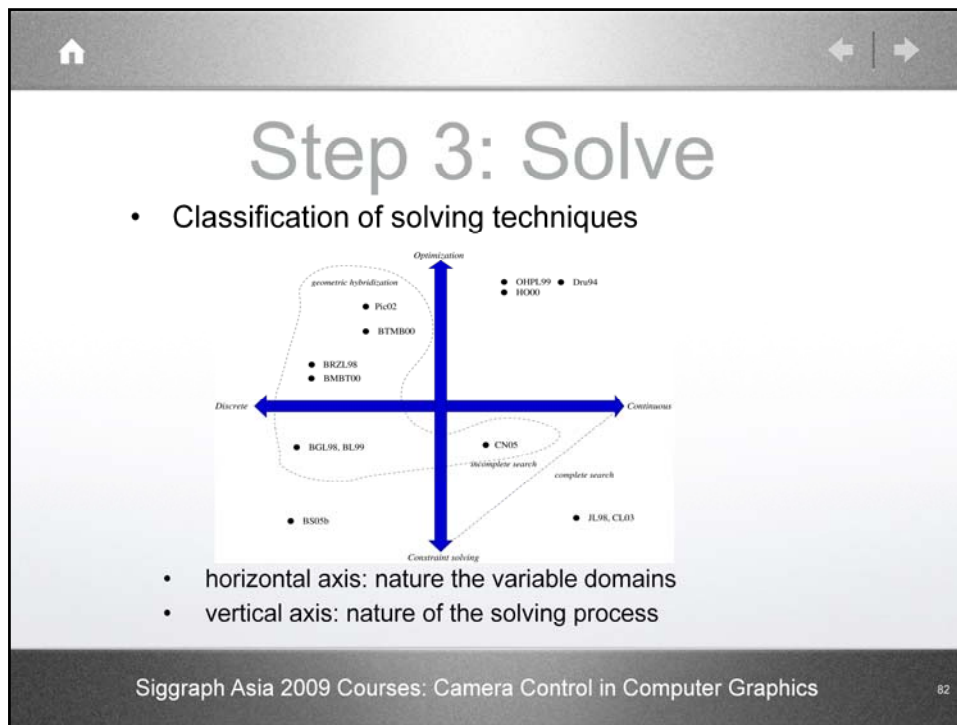
Need to constrain an object s to a rectangular frame on the screen:

- checking the projection of a single point: projection is a strongly non-linear relation
- cannot be repeated for every point: issue of the representation of the object

Bounding spheres: not appropriate if no hierarchical representation is provided

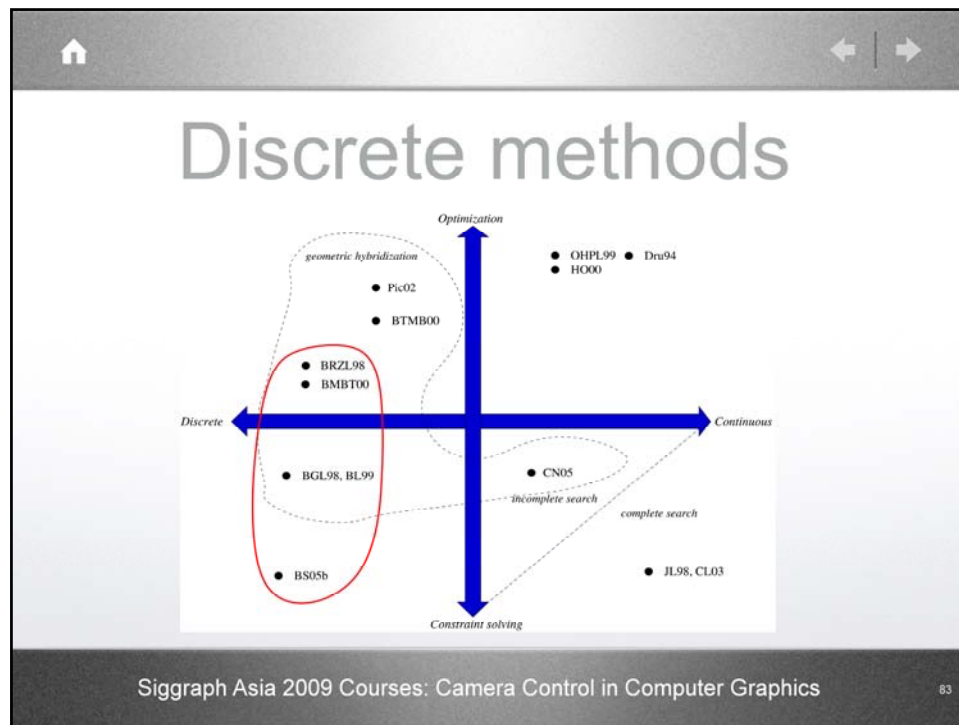
Bounding boxes: requires the projection of all 8 vertices (can be simplified by only considering the extreme vertices)

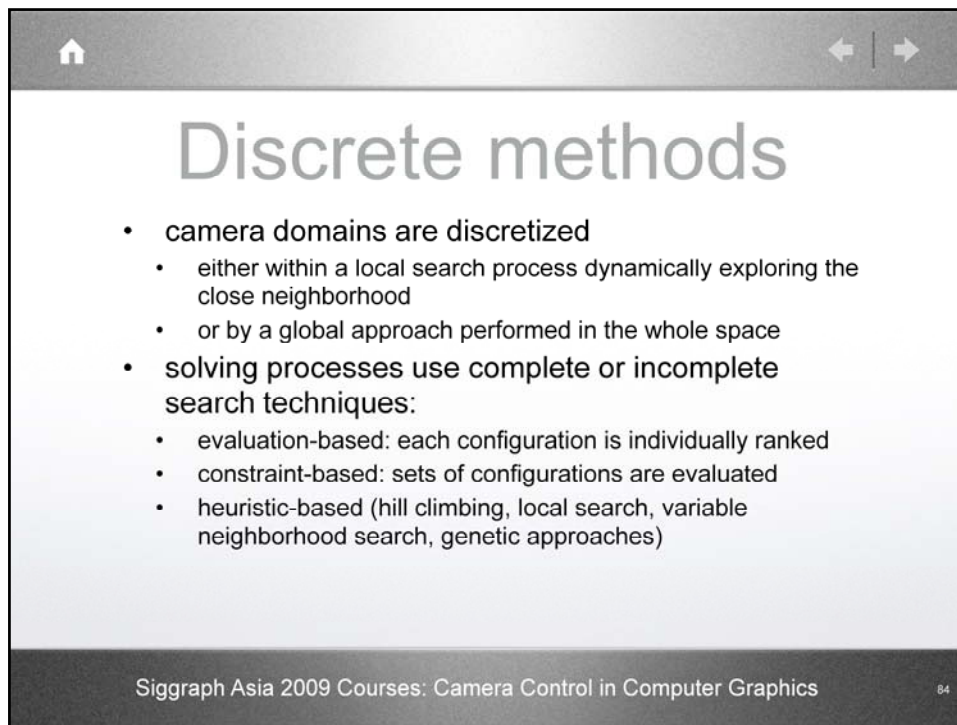
Exact representations can be used by performing some hardware rendering and checking pixel overlap with the frame, but difficult to compute the derivatives of such functions when using optimisation-based packages as solvers.



Classification of approaches in the literature

- on the horizontal axis, there is a progression from left to right, between discrete and continuous methods. Discrete methods consider a (regular or non regular) sampling of the search space either locally around a configuration, or globally. Continuous methods consider the full range of real values (as would do gradient-based optimizers)
- on the vertical axis, there is a progression from pure constraint-based methods to pure optimization-based methods. In constraint-based methods, each property is expressed as a (hard) constraint and a solution should satisfy all the constraints simultaneously. In optimization-based methods, each property is expressed as a fitness function. Functions are then individually aggregated into a function to minimize (maximise)



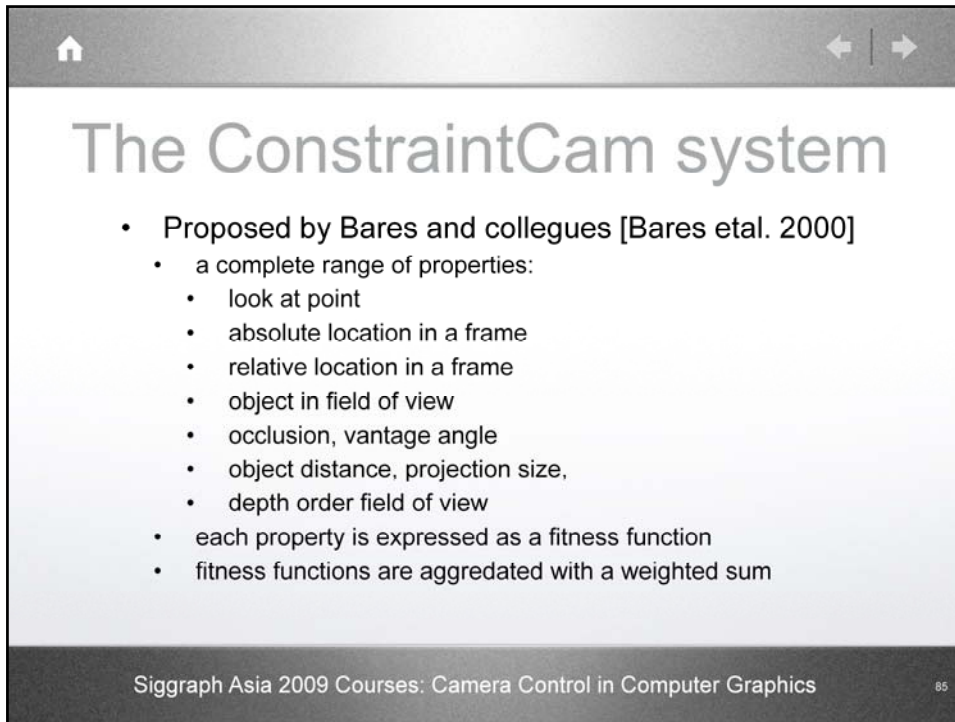


One can further classify the discrete techniques according to their complete or incomplete nature: are all the camera configurations evaluated or not?

- in complete techniques, the whole search space is sampled for camera configurations. Not really appropriate for dynamic environments (or configurations need to be re-evaluated)
- in incomplete techniques, the space is either locally sampled around a current configuration (in a way inspired from local search techniques), or refined in promising areas

Solving:

- evaluation-based: expensive, misses potential solutions
- constraint-based: some techniques (interval techniques [Moore66]) allow to (over) evaluate a whole range of solutions, which enables pruning to be performed
- heuristic-based: classical hill-climbing, branch and bound, etc.. don't guarantee the optimality, cost can be easily tuned.

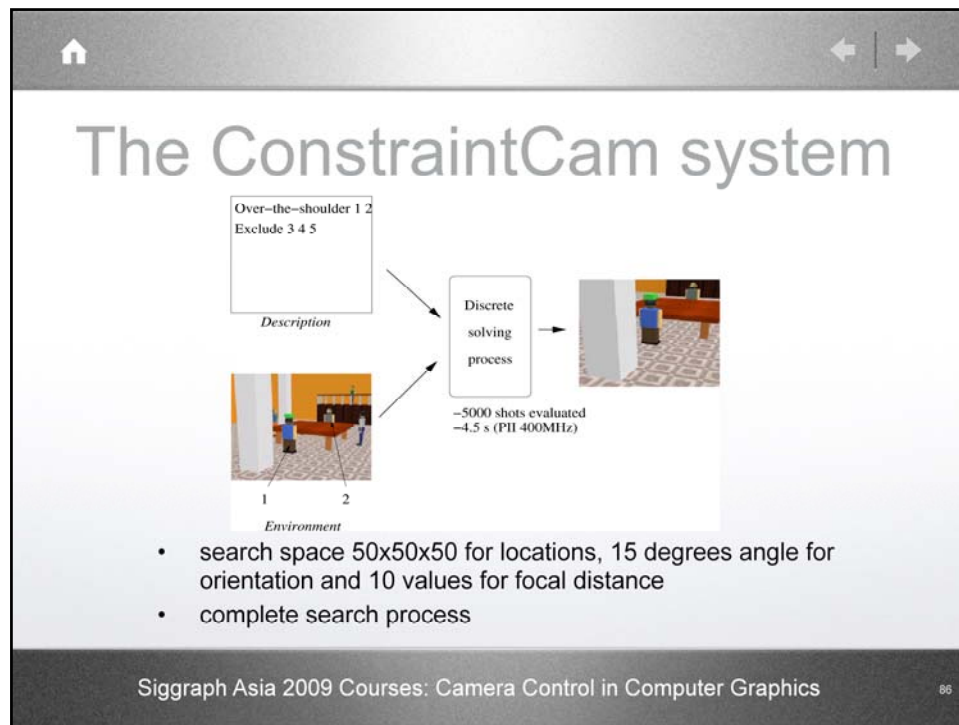


The ConstraintCam system

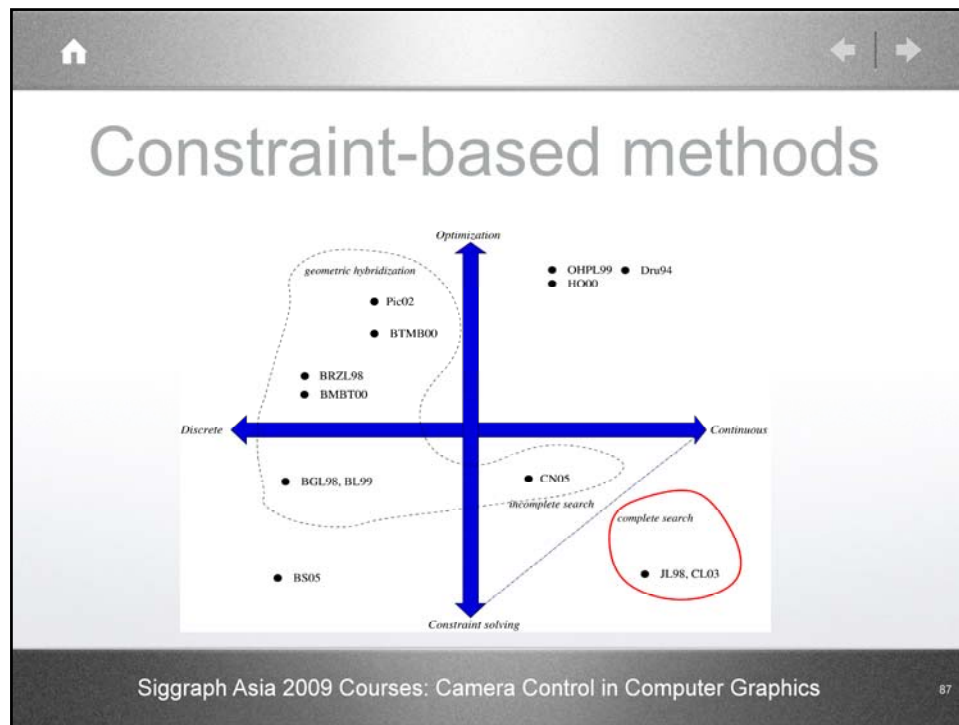
- Proposed by Bares and colleagues [Bares et al. 2000]
 - a complete range of properties:
 - look at point
 - absolute location in a frame
 - relative location in a frame
 - object in field of view
 - occlusion, vantage angle
 - object distance, projection size,
 - depth order field of view
 - each property is expressed as a fitness function
 - fitness functions are aggregated with a weighted sum

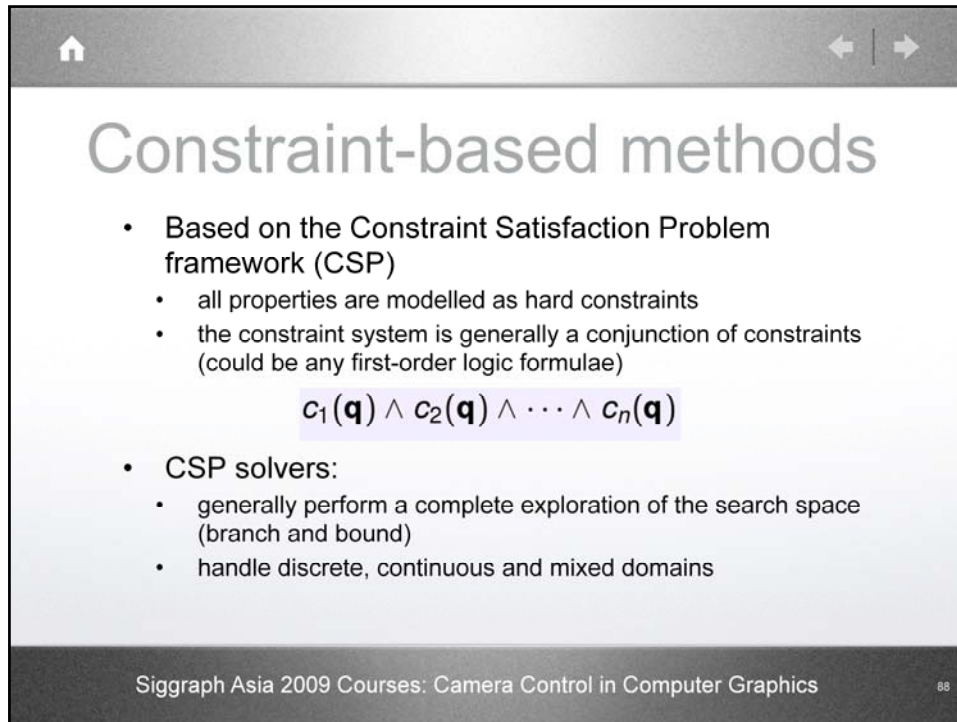
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 85

ConstraintCam considers the problem of viewpoint computation for composition. The authors propose a large set of properties, including occlusion and rectangular bounding for targets (complex properties to evaluate). Discretization steps are quite large, but due to the number of dimensions to explore (7), a very large number of configurations is explored (> 13M). Improvements proposed later by the authors use a refinement process around “interesting” regions (ie. where the fitness functions provide better values).



Displays an example of the ConstraintCam system using a high-level description (e.g. over the shoulder shot involves 5 to 6 functions). Properties can be emphasized by adding a degree of preference expressed as a scalar value (generally normalized).





The slide is titled "Constraint-based methods" and features a list of bullet points. The first bullet point is "Based on the Constraint Satisfaction Problem framework (CSP)", which has two sub-bullets: "all properties are modelled as hard constraints" and "the constraint system is generally a conjunction of constraints (could be any first-order logic formulae)". Below these is the formula $c_1(\mathbf{q}) \wedge c_2(\mathbf{q}) \wedge \dots \wedge c_n(\mathbf{q})$. The second main bullet point is "CSP solvers:", with sub-bullets: "generally perform a complete exploration of the search space (branch and bound)" and "handle discrete, continuous and mixed domains". The slide has a dark header with a home icon and navigation arrows, and a dark footer with the course name and page number 88.



Constraint-based methods

- Based on the Constraint Satisfaction Problem framework (CSP)
 - all properties are modelled as hard constraints
 - the constraint system is generally a conjunction of constraints (could be any first-order logic formulae)
$$c_1(\mathbf{q}) \wedge c_2(\mathbf{q}) \wedge \dots \wedge c_n(\mathbf{q})$$
- CSP solvers:
 - generally perform a complete exploration of the search space (branch and bound)
 - handle discrete, continuous and mixed domains

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 88

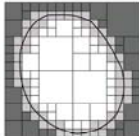
Constraint-based techniques rely on the expression of each cinematographic property as a (hard) constraint, i.e. that is either fully satisfied or not satisfied.

Since the domains of the camera parameters are generally large (with a large discrete set or a continuous set), the solving processes will compute a set of solutions satisfying all constraints simultaneously (or satisfying most of the constraints when over-constrained), instead of a single solution maximizing the aggregation of fitness functions (as with classical optimization schemes)



Constraint-based methods

- Using interval arithmetic [JL98, CL03, CLB09]
 - the problem variables are modelled as intervals of floating point values
 - intervals compute a partitioning of the search space into fully-satisfied areas, partially-satisfied areas and non-satisfied areas
- efficient extensions (see <http://sourceforge.net/projects/realpaver/>)



- contains no solutions
- contains possible solutions
- contains only solutions



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics89

Interval-based techniques employ a specific representation in which domains of the camera parameters are modelled as an interval of floating-point values. All computations are therefore performed over these interval values. The solving process partitions these domains into three categories:

- domains that are fully satisfied (i.e. every floating point value in the domain is a solution to the set of constraints)
- domains that not satisfied (i.e. every floating point value in the domain is not a solution of the whole set of constraints – but could be a solution to a subset of constraints)
- domains that are partially satisfied (i.e. the domains possibly contains both solutions and non-solutions, and must be further bisected).

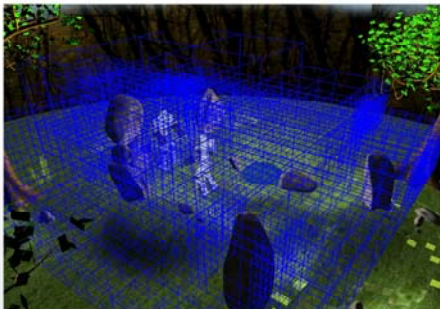
The process iterates partitions over the domain up to a minimal splitting size. Recent techniques (using Newton-based interval computations) implement efficient pruning operators that reduce the practical complexity.

Closely related to kd-trees (the satisfaction is computed for every node)



Interval-based techniques



- Compute all the viewpoints such that "A is visible from the Front, B from $\frac{3}{4}$ rear, and B does not occlude A for time interval $[0..5]$ "



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics90


An example from [CLB09] of an interval-based technique. The description typically represents an over the shoulder shot in a face to face discussion between two characters. The interest in computing the whole partitioning of the space lays in that adjacent cells can be linked together in an (adjacency) graph, and paths can be planned within the graph. Furthermore the discretization can be operated over the time dimension, which provides in turn a spatio-temporal adjacency graph (see [CLB09]).

This sequence of shots illustrates the refinement of properties in the solving process. The first shot displays collision (ie pruning partitions that collide with elements in the scene)





Interval-based techniques

- Compute all the viewpoints such that "A is visible from the Front, B from $\frac{3}{4}$ rear, and B does not occlude A for time interval $[0..5]$ "




Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 91

Second shot displays front wedge for the first character (A is visible from the front)





Interval-based techniques

- Compute all the viewpoints such that "A is visible from the Front, B from $\frac{3}{4}$ rear, and B does not occlude A for time interval $[0..5]$ "




Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 92

Third shot show all configurations remaining where B is seen from the $\frac{3}{4}$ rear viewpoint (no side is chose according to the line of interest defined by the two actors).



Interval-based techniques

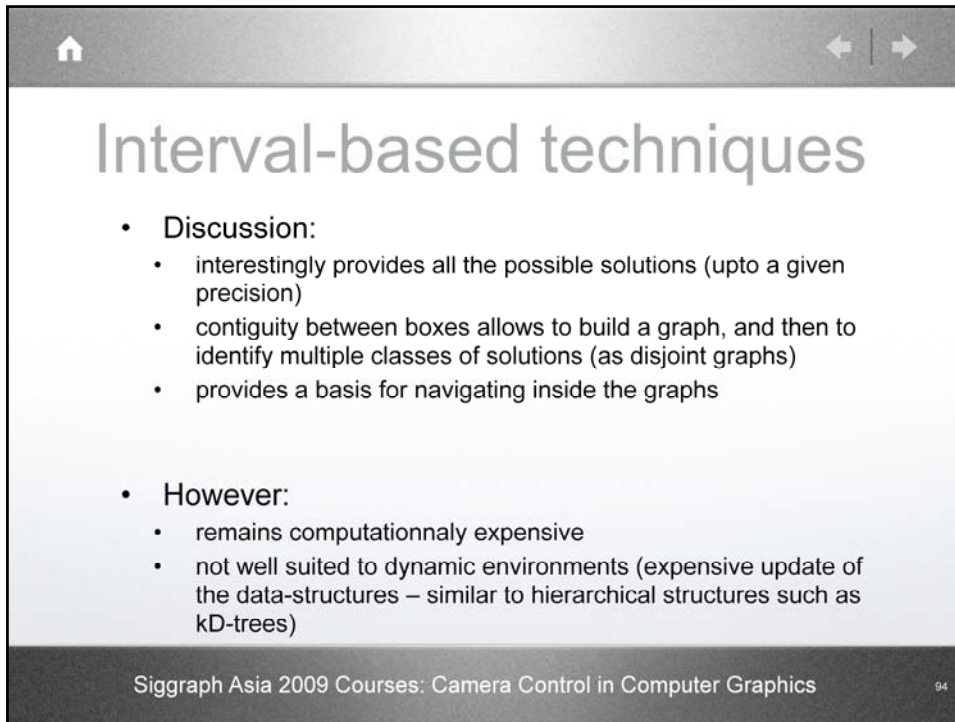
- Compute all the viewpoints such that "A is visible from the Front, B from $\frac{3}{4}$ rear, and B does not occlude A for time interval $[0..5]$ "



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics93

Last shot displays all configurations where B does not occlude A in the shot. Here shots display the computation at time $t=0$.

Computing the configurations for a discretized set of time intervals over 0 to 5 seconds, leads to the construction of an adjacency graph that evolves in time. By reasoning over the evolution of this adjacency graph in time, one can plan cinematographic shots and edits (eg. Is there a static shot such that all properties are satisfied over time interval $[0..5]$? Is there a continuous motion of the camera such that all properties are satisfied over time interval $[0..5]$? Is it possible to perform a continuous travelling for $[0..5]$?...)



Interval-based techniques

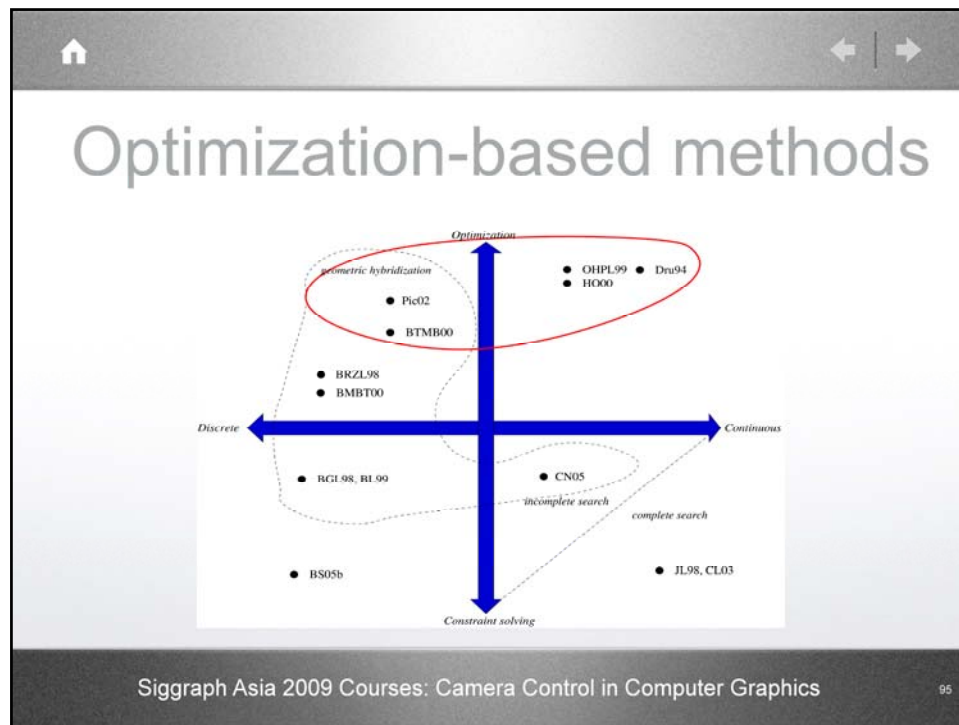
- Discussion:
 - interestingly provides all the possible solutions (upto a given precision)
 - contiguity between boxes allows to build a graph, and then to identify multiple classes of solutions (as disjoint graphs)
 - provides a basis for navigating inside the graphs
- However:
 - remains computationnaly expensive
 - not well suited to dynamic environments (expensive update of the data-structures – similar to hierarchical structures such as kD-trees)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 94

•Contiguity: by maintaining the contiguity information between the boxes in the solving process, one can build a graph (each box is a node, two contiguous boxes build an arc). Some reasoning steps can be performed on the graphs (identification of bottle-necks, identification of disjoint sub-graphs, depth which provides an idea of how robust the area is...)

•Navigation: the graph structure is then well suited for navigation processes, and some properties can be dynamically recomputed if necessary.

•Expensive process: obviously exponential in the number of camera parameters. Classically contributions generally split into two: (i) the one of computing the camera location and (ii) the one of computing the camera orientation.





Shots are expressed as a set of functions to optimize (classically a linear combination of individual fulfilments), and compared to constraint-based approaches will compute a single solution to the problem. The first contribution to the domain was proposed by S. Drucker, who solved cinematographic properties by using Sequential Quadratic Programming (function is locally approximated by a second degree equation). SQP techniques typically suffer from local minimas and are highly dependent on the starting configuration.

Optimization-based methods

- properties are modelled as a set of functions to minimize/maximize: fitness functions $f_i(\mathbf{q})$
- general expression on the problem:
$$\text{minimize } F(f_1(\mathbf{q}), f_2(\mathbf{q}), \dots, f_n(\mathbf{q})) \text{ s.t. } \mathbf{q} \in \mathbf{Q}.$$
- the aggregation function F is generally expressed as a sum of scalar-weighted normalized functions:
$$F(f_1(\mathbf{q}), f_2(\mathbf{q}), \dots, f_n(\mathbf{q})) = \sum_{i=1}^n w_i f_i(\mathbf{q})$$
- Solving techniques:
 - deterministic: gradient-based, quadratic programming,...
 - stochastic: local search, genetic algorithms, simulated annealing

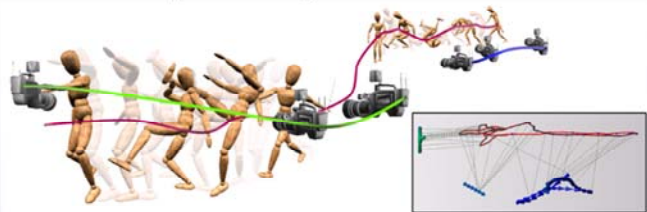
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 96

Literature in the domain of virtual camera control displays quite a large range of solving techniques. In their simplest expression, contributions rely on ad-hoc solving techniques (eg. the local search paradigm simply considers the iterative exploration of the neighborhood of a current configuration by repeating local moves towards the best neighbour, with some multiple forms of escape strategies in case of a supposed local minima). Stochastic techniques (eg. genetic algorithms, simulated annealing) are less prone to local minima and less dependent of initial configurations, but in general remain computationally expensive.



Optimization-based methods

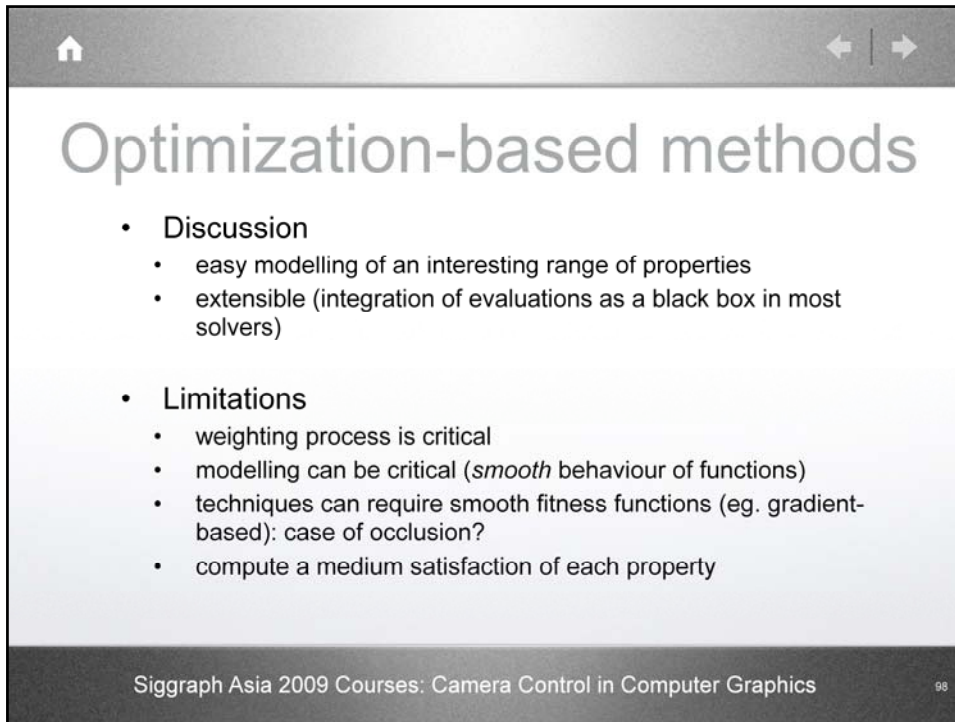
- Assa et al. [ACYL08] compute camera viewpoints/paths that emphasize the motions of human actions.
 - maximize the viewpoint quality by considering a linear combination of character's motion, orientation and human-pose
 - optimization process can split the path into multiple pieces
 - path construction is performed by local moves



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 97

Assa et al. rely on a number of descriptors employed in viewpoint computation (silhouette, frame entropy, ...). For the silhouette, the authors propose to study the pose joint locations as a point cloud, and extract the plane with the largest projection (using a Principal Component Analysis). Other properties they consider are: movement direction of the actor, facing the character, widest aspect and visibility of the limbs (computed by hardware renderings). Saliency of motions is computed by comparing a low dimensional curve representation of relative joint location and speed, with a smooth average curve. Computation is offline.

Paths are built by locally exploring a neighborhood around the current configuration, following the best viewpoint evaluation. At each step, the relative quality of the viewpoint is computed (ratio of the local solution and the best global solution). This metric is used to split the path as soon as the ratio reaches a given threshold (70%).



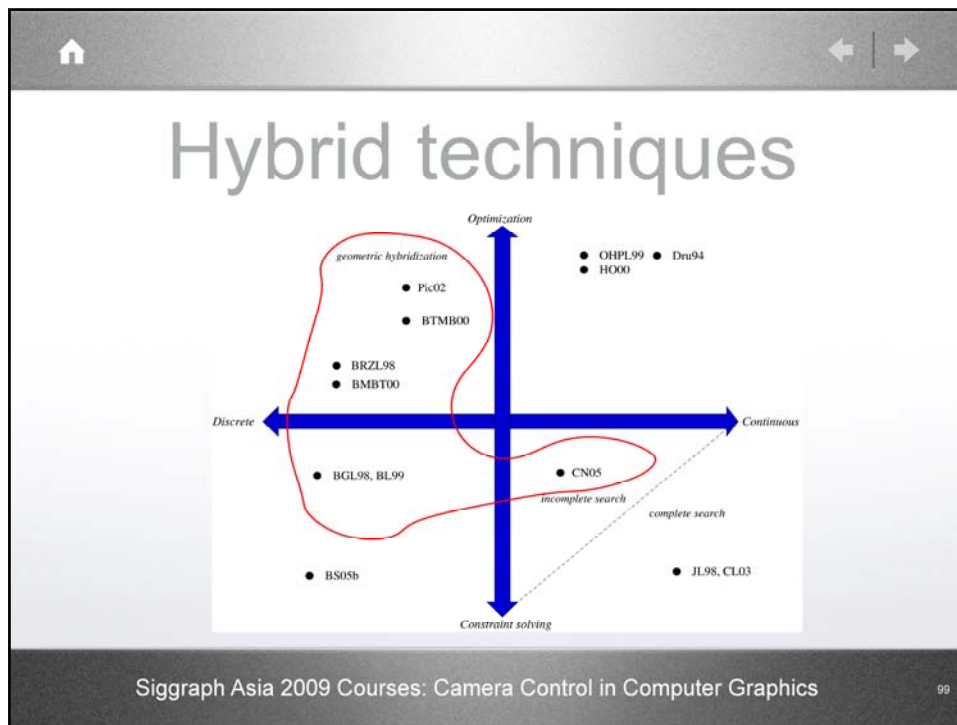
Optimization-based methods

- Discussion
 - easy modelling of an interesting range of properties
 - extensible (integration of evaluations as a black box in most solvers)
- Limitations
 - weighting process is critical
 - modelling can be critical (*smooth* behaviour of functions)
 - techniques can require smooth fitness functions (eg. gradient-based): case of occlusion?
 - compute a medium satisfaction of each property

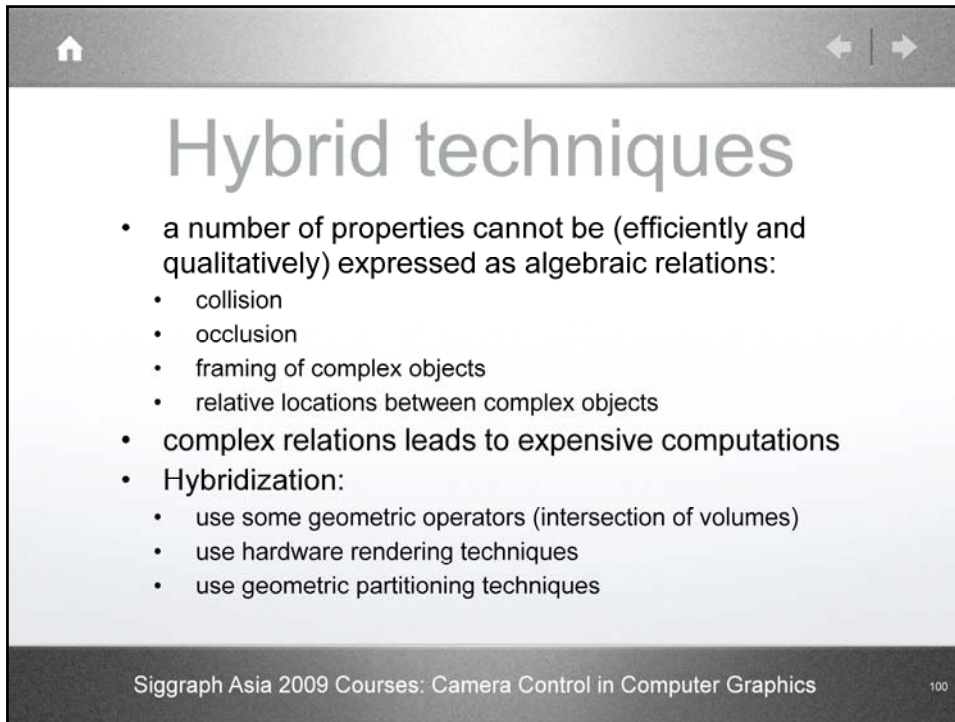
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 98

Optimization techniques offer a valuable framework for camera control: properties can be expressed in multiple ways, preferences can be added and there is a large range of solving procedures. However the complexity of 3D scenes inevitably requires to use abstractions of the underlying geometry that lead to approximations in the results (either over or under). Improving the quality of abstraction of the objects is a difficult task, which generally leads to significant computational costs. Relying on hardware rendering techniques to estimate the fulfilment of properties is a good qualitative move. However a number of solving techniques require the expression of properties as algebraic relations with smooth behaviour or ability to analytically compute the derivative – which is not compatible with rendered-based estimation.

Contributions have therefore been exploring the possibilities given by interleaving and hybridizing different solving techniques.



Hybrid techniques cover a large range of approaches on both axes. Most approaches mix geometric operators (to reduce the size of the search space) with numeric solvers (to concentrate on small search space). In that, hybrid techniques are constrained-optimization approaches (geometric operators are constraints, and properties to be fulfilled are expressed as fitness functions). Furthermore hybrid techniques are well suited to hardware rendering-based estimations.



Hybrid techniques

- a number of properties cannot be (efficiently and qualitatively) expressed as algebraic relations:
 - collision
 - occlusion
 - framing of complex objects
 - relative locations between complex objects
- complex relations leads to expensive computations
- Hybridization:
 - use some geometric operators (intersection of volumes)
 - use hardware rendering techniques
 - use geometric partitioning techniques

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 100



•Occlusion (see section on visibility): the use of bounding volumes (such as spheres in [CN05]) provide a very rough over-estimation of visibility at a reduced cost (only a scalar product). Though hierarchical models can provide better estimations, the cost in solving increases severely.

•Framing: detection of overlaps between the projected geometry of a target (or its abstraction) and screen frames remains complex when looking for precise evaluations

•Relative location: needs to study the layout produced by the projection of both targets (or their abstractions). This leads to the same quality of estimation vs. cost debate.

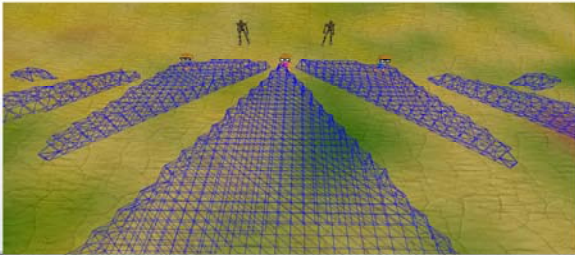
Hybridizations we review here use:

- geometric operators/partitions to reduce the size of the search space (and hence the complexity of the solving process)
- hardware rendering to improve the quality/cost ratio
- mix both (eg. Burelli et al. [BGER08])



Hybrid techniques

- Hybrid geometric/numeric [Pic02,CN05, BGER08]
 - search space is carved out by geometric operators
 - keep areas that share similar cinematographic properties (similar to the principle of visual aspects)
 - numerical stochastic approach is computed in each area

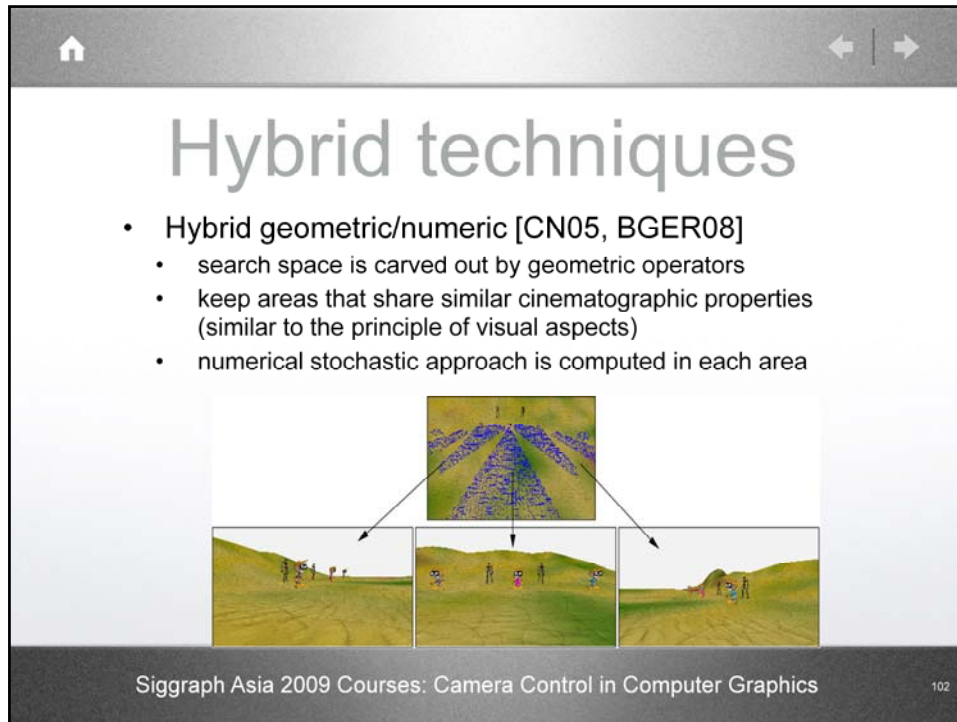


Siggraph Asia 2009 Courses: Camera Control in Computer Graphics101

In [Pic02] and [CN05], authors separate properties into those that can be expressed in terms of geometric volumes, and those that can be expressed in terms of fitness functions. For example, viewing the front of a character can be expressed by selecting a pyramid volume of camera locations (from -45 to +45 degrees around the character's front vector).

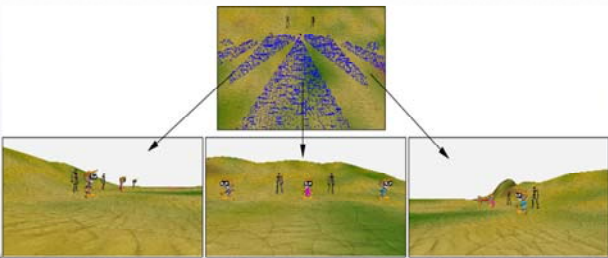
All volumes are then intersected and the search process is performed inside the remaining volume ([Pic02] use quad-trees, [CN05,BGER05] use implicit volumes). Contradictory properties can be easily detected (empty intersection), which is not the case with optimization techniques.

More precise properties such as exact screen locations for the targets, on-screen distances, percentage of overlap are easily handled by fitness functions. Other hybridizations are possible, as in [BGER08] which employ geometric operators, and ray-casting for evaluating occlusion.



Hybrid techniques

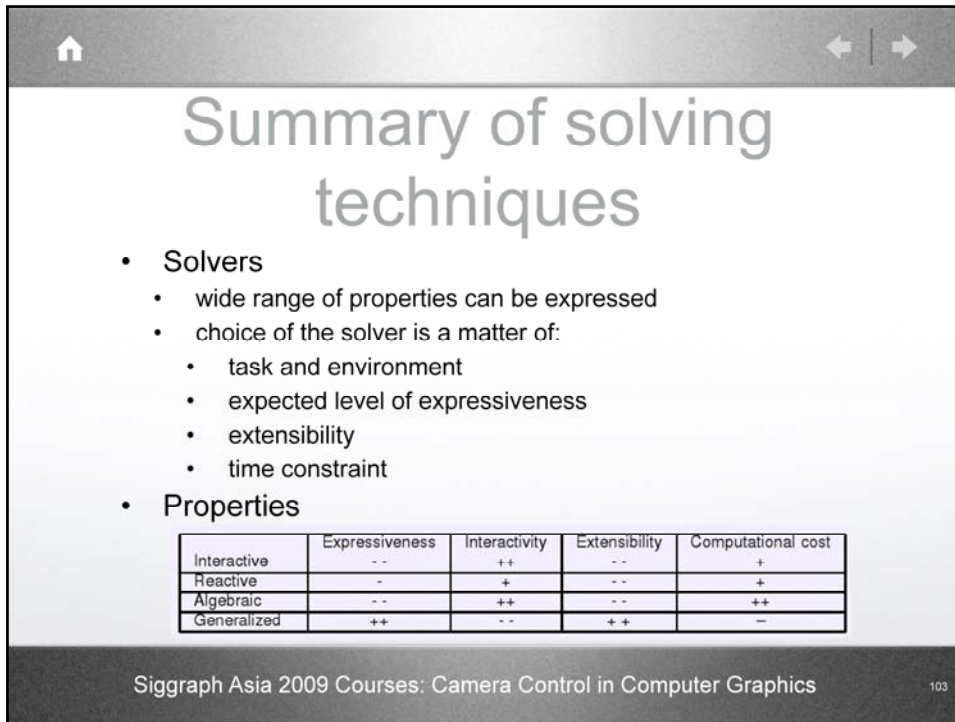
- Hybrid geometric/numeric [CN05, BGER08]
 - search space is carved out by geometric operators
 - keep areas that share similar cinematographic properties (similar to the principle of visual aspects)
 - numerical stochastic approach is computed in each area



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 102

In [CN05], authors develop the notion of semantic volumes (an extension of visual aspects). In visual aspects, all the viewpoints of a single polyhedron that share similar topological characteristics in the projected image are enumerated, and a change of appearance of the projected polyhedron when changing the viewpoint allows to partition the viewpoint space. Furthermore these regions can be connected through adjacency to build an aspect graph. The authors extent this notion to encompass cinematographic properties. Thus viewpoints with similar cinematographic properties are gathered.

In the numeric computation of solutions, only one viewpoint per volume is computed. In the picture, each volume leads to a significantly distinct layout of elements on the screen.



Summary of solving techniques

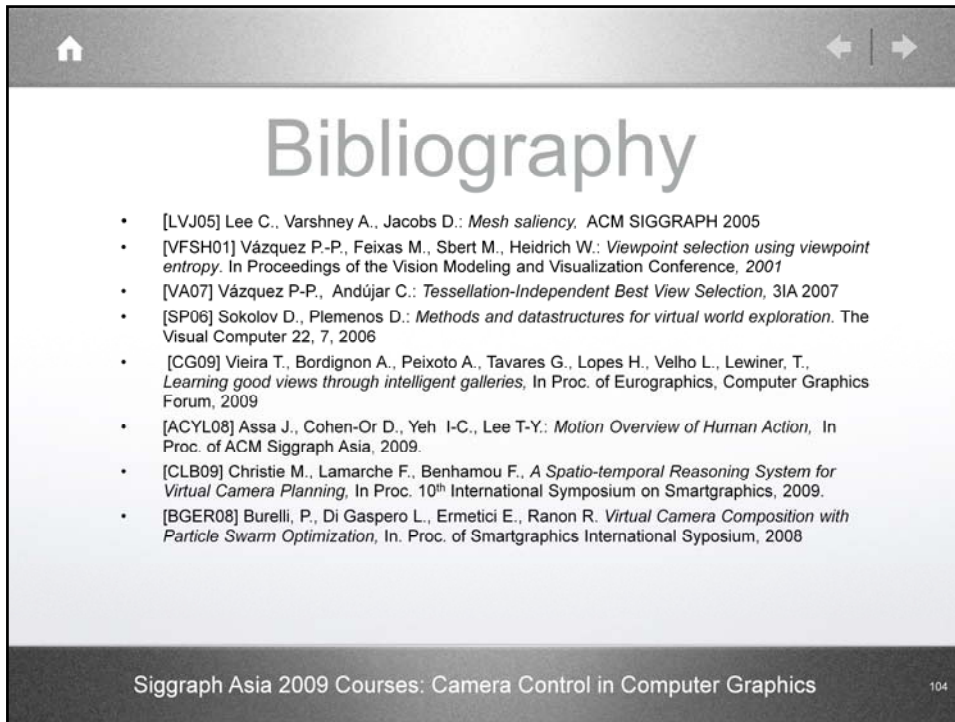
- Solvers
 - wide range of properties can be expressed
 - choice of the solver is a matter of:
 - task and environment
 - expected level of expressiveness
 - extensibility
 - time constraint
- Properties

	Expressiveness	Interactivity	Extensibility	Computational cost
Interactive	--	++	--	+
Reactive	-	+	--	+
Algebraic	--	++	--	++
Generalized	++	--	++	-

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 103

In summary, a range of solving techniques is available. Solvers differ as to how they manage over-constrained and under-constrained problem formulation, whether they perform a complete or heuristic-based exploration, how extensible they are, and their possibility to hybridize.

The table displays the properties of main solving techniques in terms of expressiveness (nature and range of properties that can be expressed), interactivity (mainly a factor of real-time computation), extensibility (how easy to integrate new properties in the solving process) and computational cost.

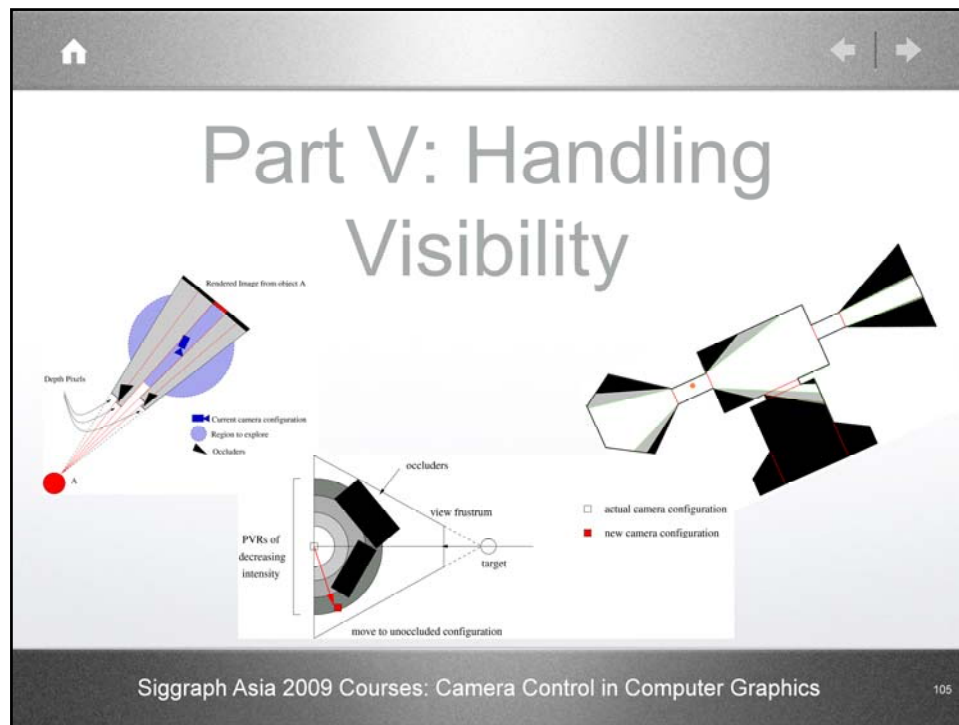


Bibliography

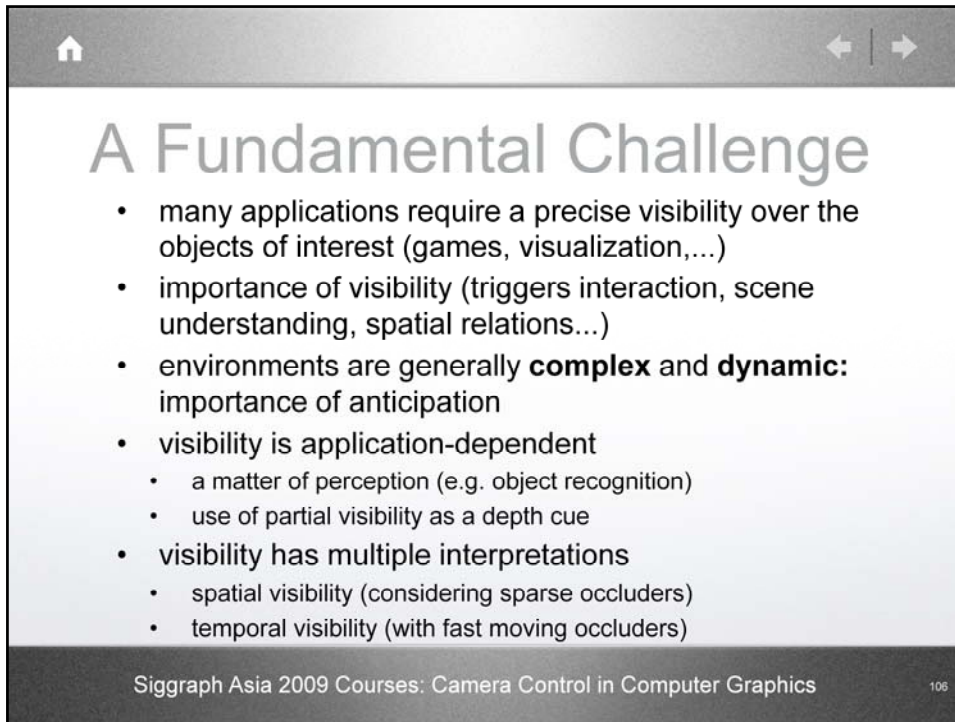
- [LVJ05] Lee C., Varshney A., Jacobs D.: *Mesh saliency*, ACM SIGGRAPH 2005
- [VFSH01] Vázquez P.-P., Feixas M., Sbert M., Heidrich W.: *Viewpoint selection using viewpoint entropy*. In Proceedings of the Vision Modeling and Visualization Conference, 2001
- [VA07] Vázquez P.-P., Andújar C.: *Tessellation-Independent Best View Selection*, 31A 2007
- [SP06] Sokolov D., Plemenos D.: *Methods and datastructures for virtual world exploration*. The Visual Computer 22, 7, 2006
- [CG09] Vieira T., Bordignon A., Peixoto A., Tavares G., Lopes H., Velho L., Lewiner, T., *Learning good views through intelligent galleries*, In Proc. of Eurographics, Computer Graphics Forum, 2009
- [ACYL08] Assa J., Cohen-Or D., Yeh I.-C., Lee T.-Y.: *Motion Overview of Human Action*, In Proc. of ACM Siggraph Asia, 2009.
- [CLB09] Christie M., Lamarche F., Benhamou F., *A Spatio-temporal Reasoning System for Virtual Camera Planning*, In Proc. 10th International Symposium on Smartgraphics, 2009.
- [BGER08] Burelli, P., Di Gaspero L., Ermetici E., Ranon R. *Virtual Camera Composition with Particle Swarm Optimization*, In. Proc. of Smartgraphics International Symposium, 2008

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

104



All camera control frameworks fundamentally rely on being able to compute and reason about the visibility of target objects in dynamic environments. Yet in contrast to shadow computation and occlusion culling, the issue of visibility in camera control has received relatively little attention. Current real-time approaches to the computation of occlusion-free views of target objects (e.g. in computer games) rely almost exclusively on simple ray casting techniques, although approaches tend to now rely on hardware rendering techniques based on shadow volume and shadow penumbra computation.



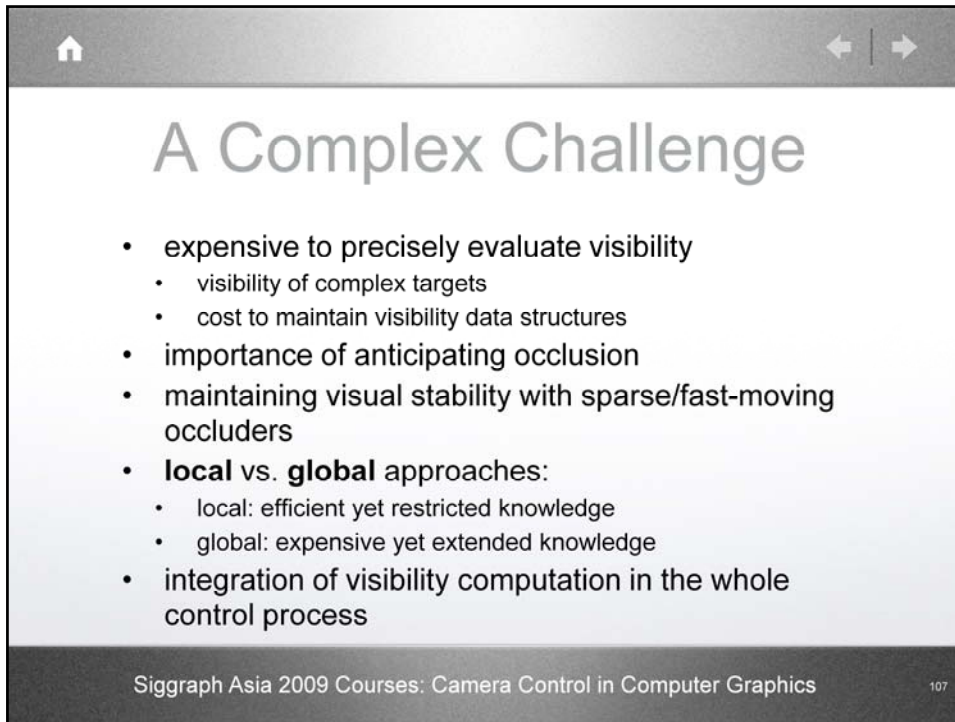
A Fundamental Challenge

- many applications require a precise visibility over the objects of interest (games, visualization,...)
- importance of visibility (triggers interaction, scene understanding, spatial relations...)
- environments are generally **complex** and **dynamic**: importance of anticipation
- visibility is application-dependent
 - a matter of perception (e.g. object recognition)
 - use of partial visibility as a depth cue
- visibility has multiple interpretations
 - spatial visibility (considering sparse occluders)
 - temporal visibility (with fast moving occluders)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 106

Visibility is a central challenge in camera control. Games, for example, require to maintain the visibility of the player and of secondary elements simultaneously (opponents, exits, items,...). Furthermore, games have been operating an important move these last years to a more cinematic experience. In scientific visualization, data may be hidden in a complex geometry setups that evolve over time. In navigation tasks, maintaining the visibility of multiple known landmarks avoids the users from getting lost or losing time in re-orientation.

However visibility is application dependent and has multiple interpretations, which means there is no generic solution to the problem. One can look at the overview proposed by [Elmqvist08] who details techniques to handle occlusion in data and object visualization (however not on how to compute viewpoints that maintain visibility, but on how to alter the geometry or scene graph). This section only considers means to evaluate occlusion and to escape from occlusion.



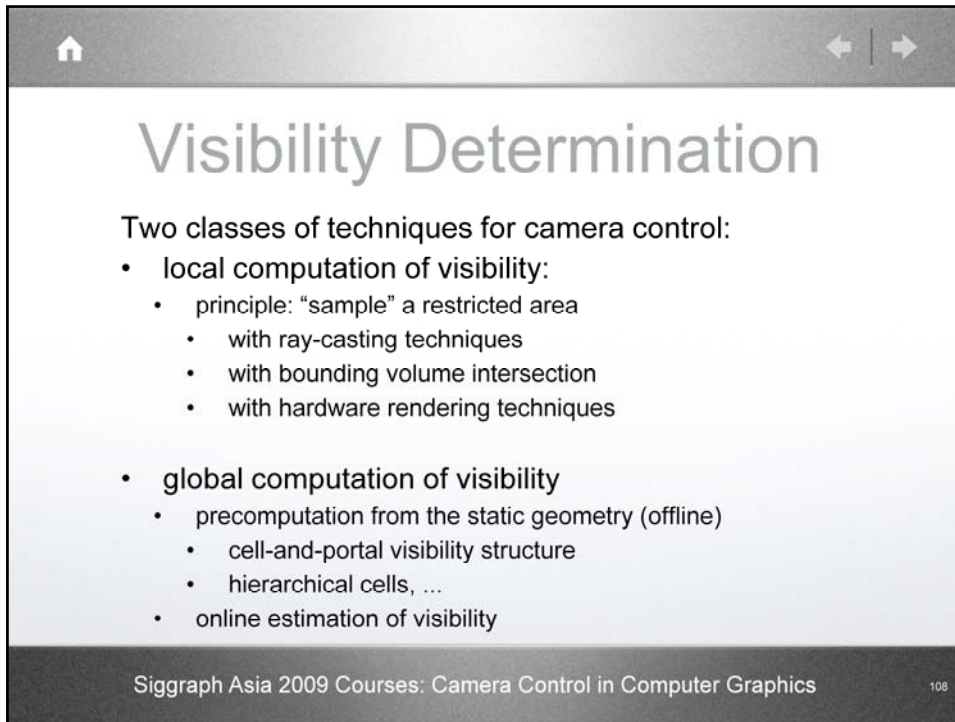
A Complex Challenge

- expensive to precisely evaluate visibility
 - visibility of complex targets
 - cost to maintain visibility data structures
- importance of anticipating occlusion
- maintaining visual stability with sparse/fast-moving occluders
- **local** vs. **global** approaches:
 - local: efficient yet restricted knowledge
 - global: expensive yet extended knowledge
- integration of visibility computation in the whole control process

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 107

The complexity of handling visibility in camera control has many sources:

- first of all, the real-time nature of most applications require efficient evaluation AND anticipation of occlusion
- second, maintaining visibility in dynamic environments is computationally expensive (as it is for occlusion culling in the field of visibility techniques for efficient rendering)
- third, the targets are generally complex-shaped objects, for which the estimation of the full visibility is an expensive process.



The slide is titled "Visibility Determination" in a large, bold, sans-serif font. Above the title is a navigation bar with a home icon on the left and left/right arrow icons on the right. Below the title, the text "Two classes of techniques for camera control:" is followed by a bulleted list. The list is divided into two main categories: "local computation of visibility" and "global computation of visibility". The "local" category includes three sub-points: "principle: 'sample' a restricted area", "with ray-casting techniques", and "with bounding volume intersection". The "global" category includes three sub-points: "precomputation from the static geometry (offline)", "cell-and-portal visibility structure", and "hierarchical cells, ...". The final sub-point under "global" is "online estimation of visibility". At the bottom of the slide, there is a footer with the text "Siggraph Asia 2009 Courses: Camera Control in Computer Graphics" and a small number "108" on the right.

Visibility Determination

Two classes of techniques for camera control:

- local computation of visibility:
 - principle: "sample" a restricted area
 - with ray-casting techniques
 - with bounding volume intersection
 - with hardware rendering techniques
- global computation of visibility
 - precomputation from the static geometry (offline)
 - cell-and-portal visibility structure
 - hierarchical cells, ...
 - online estimation of visibility

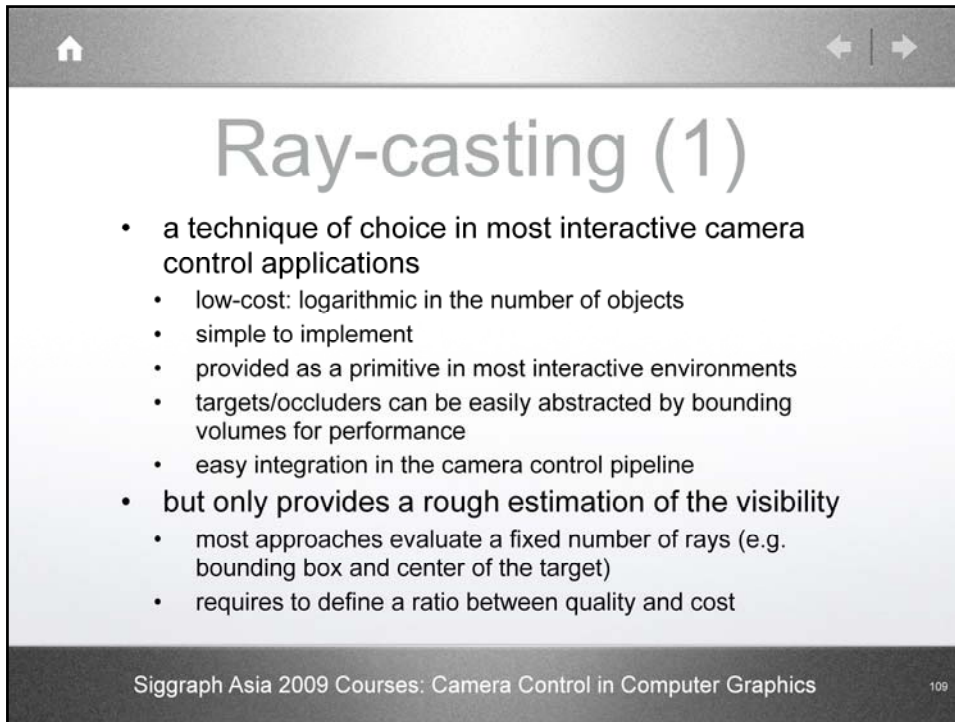
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 108

In this course, we will consider both:

- the problem of visibility determination (ie estimating how much a target is occluded)
- the problem of occlusion-free viewpoints determination (ie computing viewpoints from which target objects are visible)

For both problems, local and global techniques can be employed in similar ways:

- local techniques rely on a restricted knowledge of the environment (but can be easily updated)
- global techniques rely on a full knowledge of the visibility in the environment (that is expensive to update)

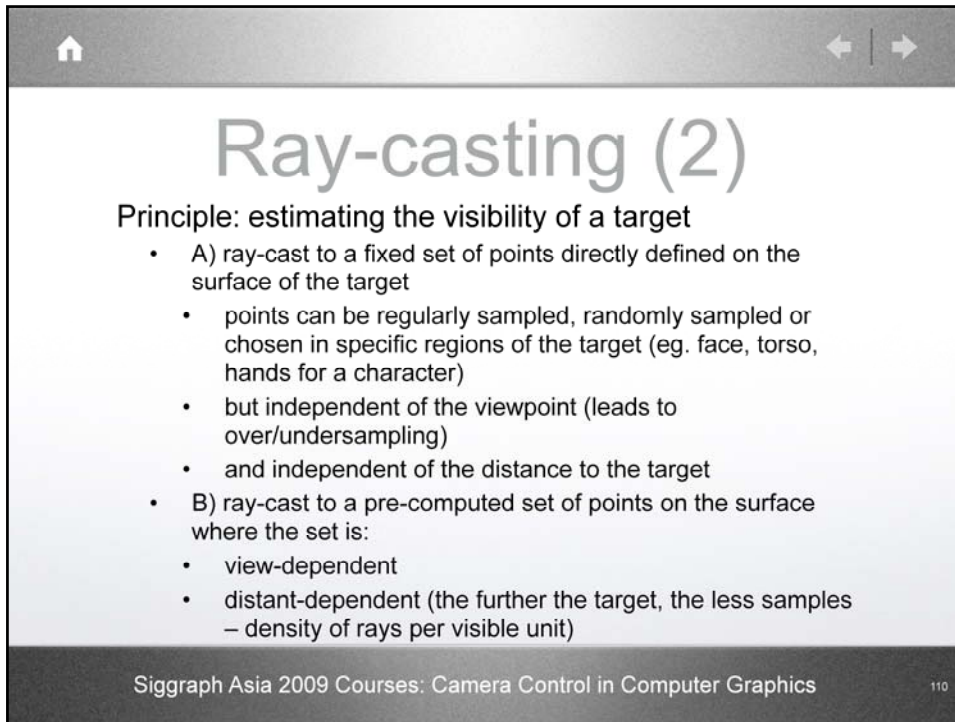


Ray-casting (1)

- a technique of choice in most interactive camera control applications
 - low-cost: logarithmic in the number of objects
 - simple to implement
 - provided as a primitive in most interactive environments
 - targets/occluders can be easily abstracted by bounding volumes for performance
 - easy integration in the camera control pipeline
- but only provides a rough estimation of the visibility
 - most approaches evaluate a fixed number of rays (e.g. bounding box and center of the target)
 - requires to define a ratio between quality and cost

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 109

In ray casting approaches the candidate position for the camera is evaluated by casting a ray in the direction of the target object. An incremental improvement on simple ray casting approaches can be achieved by casting from an array of candidate camera locations (at a linear increase in cost), and, where the visibility of multiple target objects is required, by repeating the process for each target object. Deciding how to move the camera based on such collections of single point estimates of visibility has a number of limitations, for example, it is not possible to maintain partial visibility of a target object as it moves behind a sparse occluder (such as a set of railings). Furthermore, using a single point to approximate the geometrical complexity of a target object fails to sufficiently characterize its visibility.



The slide is titled "Ray-casting (2)" in a large, bold, sans-serif font. Below the title, the text "Principle: estimating the visibility of a target" is displayed. This is followed by a bulleted list of two main methods, A and B, each with sub-points. The slide has a dark grey header bar with a home icon on the left and navigation arrows on the right. The footer contains the course name and the slide number 110.

Ray-casting (2)

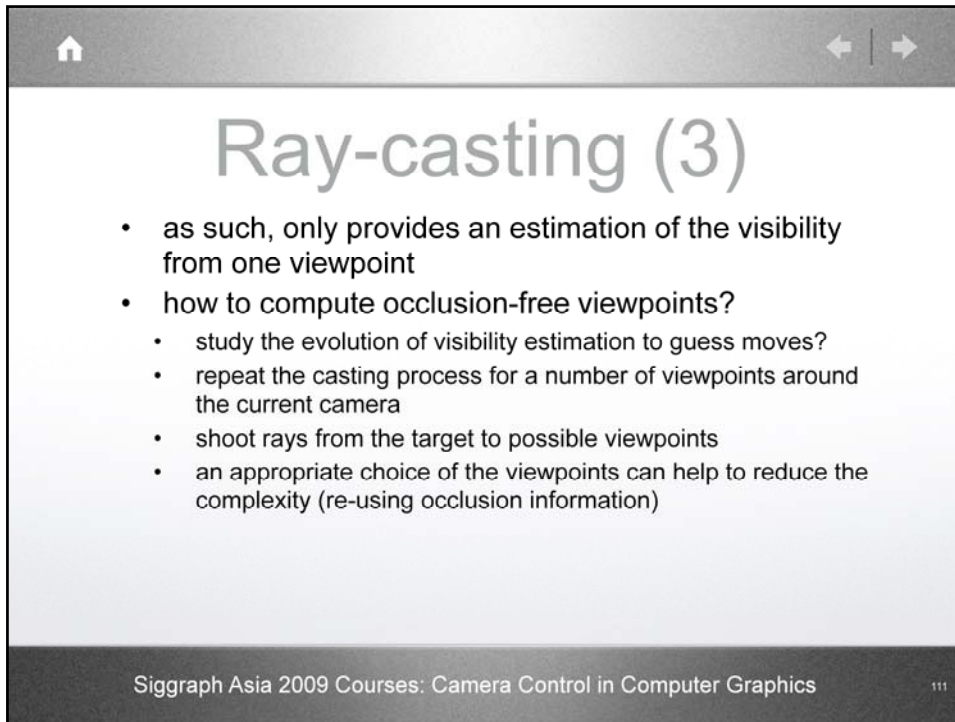
Principle: estimating the visibility of a target

- A) ray-cast to a fixed set of points directly defined on the surface of the target
 - points can be regularly sampled, randomly sampled or chosen in specific regions of the target (eg. face, torso, hands for a character)
 - but independent of the viewpoint (leads to over/undersampling)
 - and independent of the distance to the target
- B) ray-cast to a pre-computed set of points on the surface where the set is:
 - view-dependent
 - distant-dependent (the further the target, the less samples – density of rays per visible unit)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 110

To estimate the visibility of a target objects, rays are casted from the current camera to the surface of the target (or to its approximated surface). For example, a number of approaches to automated camera control compute the visibility over the vertices of target bounding box [BGL98, LC08, BGER08].

Since ray-casting undersamples the full extend of the target geometry, visibility is either over-estimated or underestimated. Increasing both the numbers of rays and the precision of the approximated surface improve the results. One appropriate technique consists in pre-computing a set of points on the surface of the target object, dependent of the orientation and the size of the target to improve the quality of results wrt. distance with the targets.





Ray-casting (3)

- as such, only provides an estimation of the visibility from one viewpoint
- how to compute occlusion-free viewpoints?
 - study the evolution of visibility estimation to guess moves?
 - repeat the casting process for a number of viewpoints around the current camera
 - shoot rays from the target to possible viewpoints
 - an appropriate choice of the viewpoints can help to reduce the complexity (re-using occlusion information)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 111

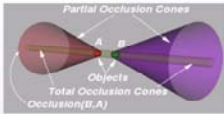
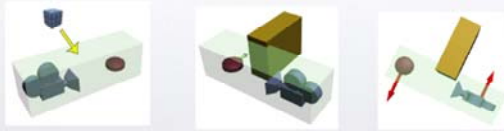
Moving from visibility estimation to the computation of occlusion-free viewpoints with ray-casting techniques.

The obvious solution consists in repeating the ray-casting process for multiple candidate viewpoints. To ensure enough diversity in the search process, a large range of viewpoints is necessary (increasing the cost). Other techniques consider casting rays from the surface of the targets to a range of viewpoints; cost is similar to the previous version, yet by appropriately choosing the viewpoints, one can reduce the cost (once a ray hits an occluder, all the configurations behind the hit are occluded, and all in front are visible).



Bounding volumes



- Principle:
 - targets and occluders are bounded by primitive shapes (spheres, AA-boxes, OBB)
 - visibility regions are built from these primitive shapes
 - intersections (or not) between primitives allow visibility detection
- Sphere-based occlusion detection [CN05]
- Axis aligned occlusion detection [MC02]



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 112

Using bounding volumes for visibility detection is a rough and conservative - yet rapid - way of estimating occlusion (i.e. can be used before more expensive techniques such as hardware rendering). Many libraries provide efficient means of detecting collision with primitives, and in most cases, the process only requires a boolean result from the test (ie. not the volume, depth or point of intersection). Courty & Marchand [CM01, MC02] avoid occlusion in a target tracking problem by computing an approximate bounding volume that encompasses both the camera and the target. Occluders (i.e. not the camera or the target objects) are prevented from entering the volume corresponding to target motion or camera motion. However, the approximate nature of the bounding volumes restricts both expressiveness (e.g. quantify partial occlusion) and practical application (e.g. over-estimation for complex shapes).

A study the evolution of the depth/volume of intersection is possible to get an idea of how occlusion is evolving. This can be used in a preventive way by proposing large volumes around the camera. However, these approximations are rough and the cost of computing the intersected volume may overshadow the lightweight advantages of the technique.




Estimating visibility (1)

Use hardware rendering techniques for estimating the visibility of a target object

- Step 1: perform a rendering of the target object
- Step 2: perform a rendering of the occluders
- Step 3: use a stencil buffer, occlusion queries, or pixel shaders to determine overlap vs. non overlap regions

- visibility can be weighted by the importance of parts of the targets
 - by rendering with an appropriate importance map
 - and multiplying visibility by importance



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 113

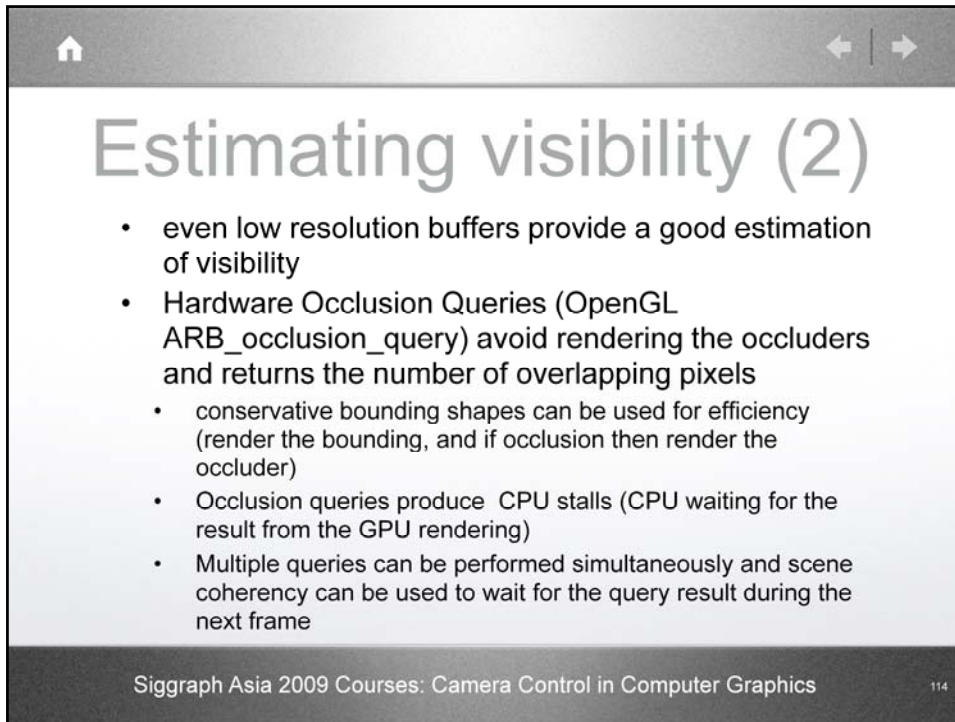
Degree of visibility of the target is determined by the ratio between the number of visible pixels of the target and the total number of pixels of the target.

Increasing the resolution of the rendered buffers obviously improves the precision in the visibility estimation (and rapidly converges to a good estimation)

Occluders and target objects can have specific geometries adapted to the rendering

- low resolution geometries, partial models (eg remove arms and legs, keep hands and feet)
- removal of sparse occluders, or alpha blended textures (e.g. fine fences, leaves etc...)

Important regions on the surface of the targets can be either manually or automatically (silhouette, saliency) computed and rendered on the surface of the target. Visibility can then be weighted by this importance map.

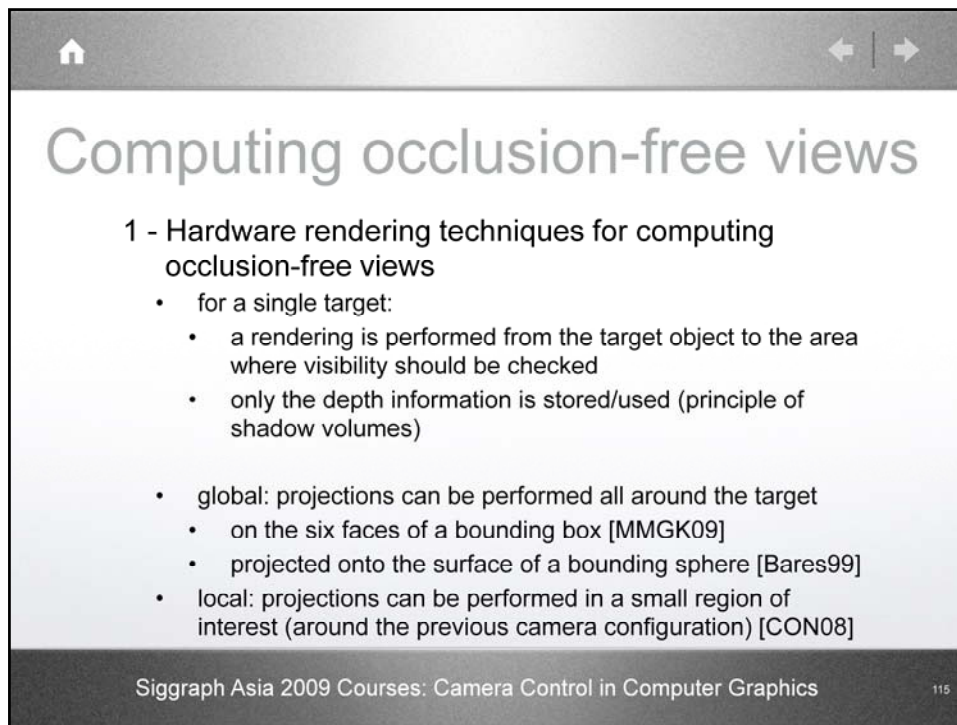


Estimating visibility (2)

- even low resolution buffers provide a good estimation of visibility
- Hardware Occlusion Queries (OpenGL ARB_occlusion_query) avoid rendering the occluders and returns the number of overlapping pixels
 - conservative bounding shapes can be used for efficiency (render the bounding, and if occlusion then render the occluder)
 - Occlusion queries produce CPU stalls (CPU waiting for the result from the GPU rendering)
 - Multiple queries can be performed simultaneously and scene coherency can be used to wait for the query result during the next frame

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 114

Using Hardware occlusion queries often lead to CPU/GPU stalls (nothing is done while waiting for the query results). An appropriate ordering of the operations must be performed while waiting for the GPU occlusion queries.



The slide is titled "Computing occlusion-free views" and is part of the "Siggraph Asia 2009 Courses: Camera Control in Computer Graphics" series. It features a navigation bar at the top with a home icon, left and right arrows, and a vertical line. The main content is a list of techniques for computing occlusion-free views, categorized under "1 - Hardware rendering techniques for computing occlusion-free views".



Computing occlusion-free views

- 1 - Hardware rendering techniques for computing occlusion-free views
 - for a single target:
 - a rendering is performed from the target object to the area where visibility should be checked
 - only the depth information is stored/used (principle of shadow volumes)
 - global: projections can be performed all around the target
 - on the six faces of a bounding box [MMGK09]
 - projected onto the surface of a bounding sphere [Bares99]
 - local: projections can be performed in a small region of interest (around the previous camera configuration) [CON08]

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 115

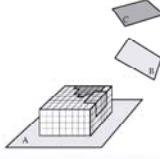
Only a small number of real-time approaches for occlusion-aware camera control have been proposed. Crucially, existing techniques (e.g [HO00]) cannot be easily extended to capture the full spatial extent of target objects (i.e. they model target objects as points). The computation of occlusion-free viewpoints is closely related to the well known problem of visibility determination [COCSD00, Dur00] which has a bearing on a range of sub-fields in computer graphics, from hidden surface removal and occlusion culling, to global illumination and image-based modeling and rendering.

Here we move from visibility estimation to the computation of occlusion-free viewpoints with hardware rendering techniques. The principle is close to the one of ray-casting: renderings are performed from the target object to the area where visibility should be checked, and most similar to the principles in shadow volume computation (studying the depth buffer to estimate whether the geometry is shadowed or not).





Computing occlusion-free views

- by projecting the occluders onto a discretized bounding volume [PBG02]
- finding a good viewpoint:
 - studying the depth map (exploring paths in the depth maps)
- extension to multiple targets is complex:
 - either build the intersection of all visible areas
 - or sample the environment and test the depths maps for each target



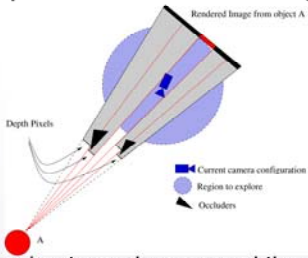
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 116

A consistent region can be computed by projecting the bounding boxes of nearby potentially occluding geometry onto a discretized sphere or box surrounding the target object, converting these projections into global coordinates, and finally negating these to yield occlusion free viewpoints [BL99, PBG92, DZ95].



Computing occlusion-free views

- By rendering from the target surface towards a regions of potential camera configurations:



- select a region to explore around the current camera
- perform the rendering
- study the depth-map for visibility

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics117

A clear parallel can be drawn between the problem of real-time soft shadow computation and real-time visibility computation of target objects. Target objects can be treated as light sources for which we need to compute the volumes outside of the shadow and penumbra (this is an inverse volume carving problem) in which to place a camera. One technique for real-time shadow computation relies on silhouette detection (e.g. penumbra wedges [AAM03]), that use the exact silhouette of objects to compute shadow volumes. However, the complexity of silhouette detection increases with the complexity the objects casting shadows and such approaches are also not readily applicable to rasterizable entities that use alpha-textures (which are increasingly used real-time 3D graphics). Another class of techniques that is used in camera control [CON08] relies on frame-buffer approaches that construct a depth map rendered from the location of light sources using graphics hardware. This shadow map is then sampled in relation to the world geometry and a simple depth comparison can be used the determine the status of a point in space (whether it is hidden by an occluder or not).

Visibility of multiple targets

- For multiple targets
 - composing visibility information: a task similar to composing multiple shadow maps (to form a penumbra map)
 - yet requires either a sampling process or the computation of the union of shadow volumes

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

In a given region, visibility for multiple targets (or multiple points on the target surface) is computed by performing one rendering per target. Depth information is composed in a way similar to penumbra maps (see next slide): the area is sampled and each sample is expressed in the local basis of each rendering in order to access the appropriate depth value in the shadow map. A specific way of composing depth maps is proposed in [CON08], where asymmetric frustums are computed for rendering. This technique avoids the sampling of the area by using a trilinear basis to access visibility information.

Visibility of multiple targets

• Visibility state for a sample:



- N : Neither of the two objects are visible
- P_A : Partially visible (only object A is visible)
- P_B : Partially visible (only object B is visible)
- V : Visible (both objects)

$$\begin{cases} N & \text{if } (z^A < z^{AI}) \wedge (z^B < z^{BI}) \\ P_A & \text{if } (z^A > z^{AI}) \wedge (z^B < z^{BI}) \\ P_B & \text{if } (z^B > z^{BI}) \wedge (z^A < z^{AI}) \\ V & \text{if } (z^A > z^{AI}) \wedge (z^B > z^{BI}) \end{cases}$$

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

Composing visibility information for a sample and two renderings: this study details the different cases. A and B are the origins of both rendered frustums, and I is the sample for which visibility is computed. Dark triangles represent occluders. In the top left frame, I is not occluded from B viewpoint, neither is it from A (the depth of the occluder being greater than the depth of the sample in A 's basis).

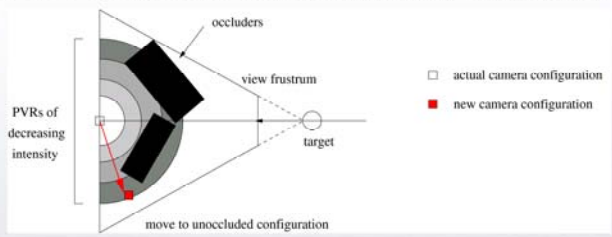
This composition maintains the knowledge of which samples are: fully occluded (from both targets), only occluded by A , only occluded by B , and fully visible. The quantitative storage enables to improve the reasoning when selecting the next location in the sample are.



Potential visibility volumes

Rendering Potential Visibility Volumes [HL01]:

- render from the target to the camera, the potential occluders (only keeping the depth information)
- build some PVS shapes at the location of the camera (a set of concentric spheres of decreasing intensity)
- render the PVS shapes by decreasing intensity with a stencil map



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

120

Hardware-based approaches to real-time visibility for camera control [HO00] evaluate the degree and extent of occlusion by rendering a scene in stencil buffers using a color for each object. Such techniques have a number of attractive properties including an independence from the internal representation of the objects, and, by avoiding bounding volumes and other geometric approximations of the object, a more accurate calculation of occlusion. Approaches based on rendering also allow the use of low resolution buffers where appropriate.

[HL01] propose to render Potential Visibility Volumes from the target viewpoint. The PVS are built as concentric shapes of decreasing intensity (using a stencil buffer to keep only the brighter ones (see next slide).



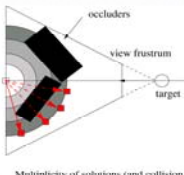
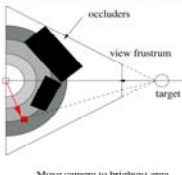
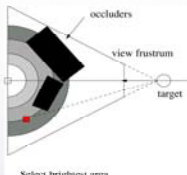



Image-based techniques

- Study the luminance in the rendered buffers:
 - the brightest areas are the best locations for the camera
 - a reverse-project from the z-buffer provides the closest unoccluded view



Select brightest area Move camera to brightest area Multiplicity of solutions (and collision issues)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

121

Left picture displays the rendered image. The brighter the colors, the better the configuration. The authors use a reverse-projection to compute the best point in the z-buffer. Collision with occluders is not handled since the z-buffer is written over by the occluders (see the right frame).

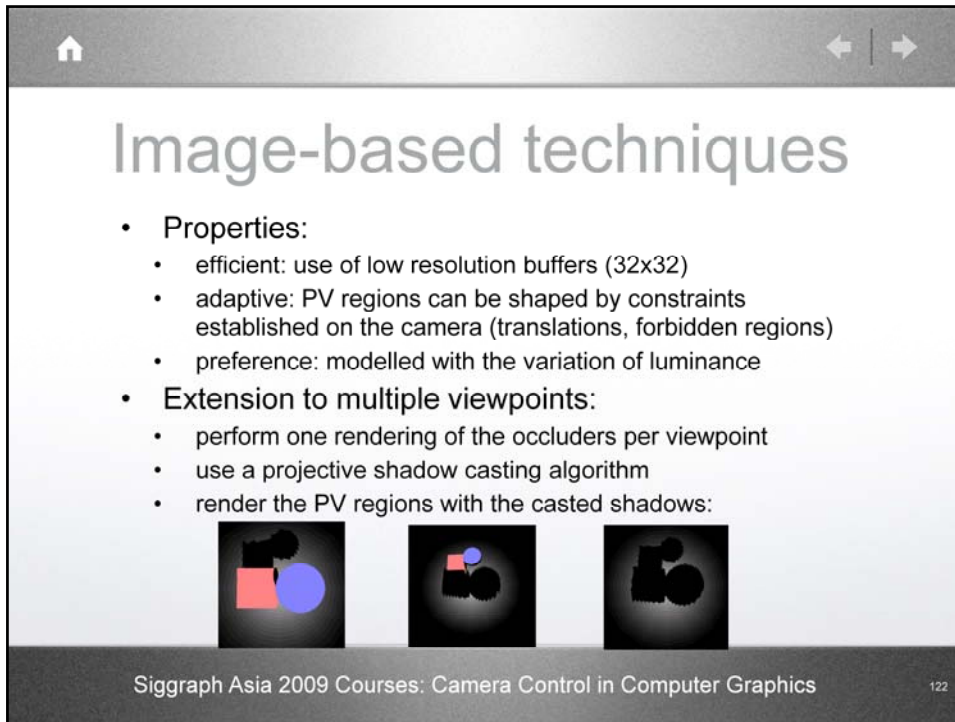



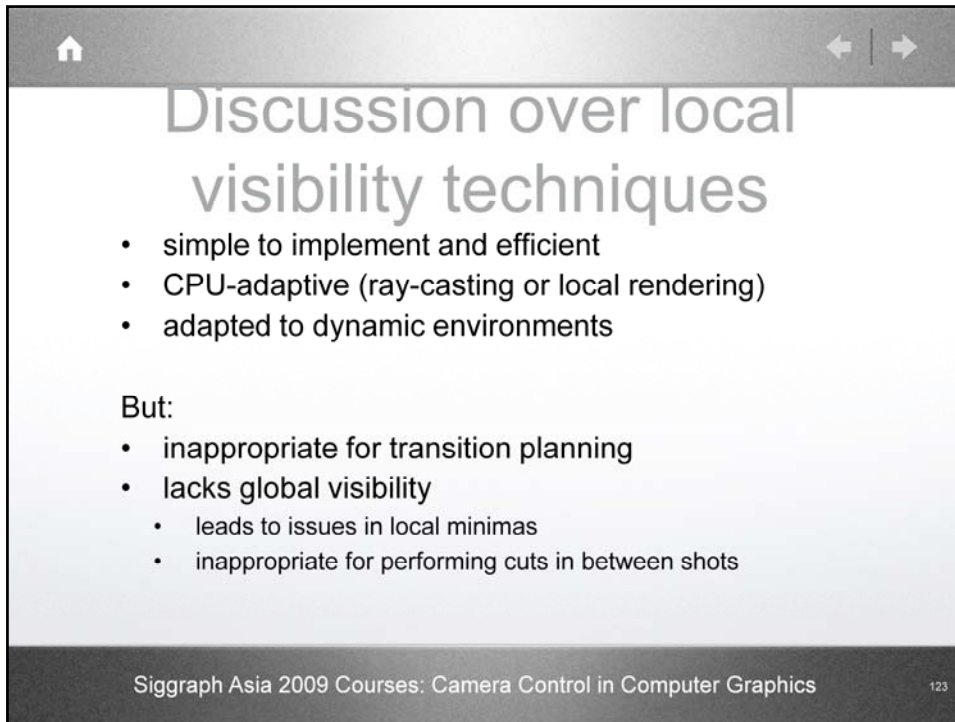
Image-based techniques

- Properties:
 - efficient: use of low resolution buffers (32x32)
 - adaptive: PV regions can be shaped by constraints established on the camera (translations, forbidden regions)
 - preference: modelled with the variation of luminance
- Extension to multiple viewpoints:
 - perform one rendering of the occluders per viewpoint
 - use a projective shadow casting algorithm
 - render the PV regions with the casted shadows:



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 122

For multiple targets, [HL01] propose to use a projective shadow casting algorithm for each viewpoint. The PV regions are then rendered from a new viewpoint, by projecting the shadows onto the surface to determine the visibility.



Discussion over local visibility techniques



- simple to implement and efficient
- CPU-adaptive (ray-casting or local rendering)
- adapted to dynamic environments

But:

- inappropriate for transition planning
- lacks global visibility
 - leads to issues in local minimas
 - inappropriate for performing cuts in between shots

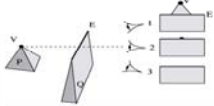
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 123

The methods we reviewed provide efficient and CPU-adaptive approaches to locally establish visibility or compute occlusion-free views. However their intrinsic local nature prevent them from performing transition planning (moving from one viewpoint to another while maximizing visibility), and may fail in some situations (no local visibility). Furthermore, when cuts between viewpoints must be computed (eg. reverse shots), many local regions need to be sampled (with no guarantee of finding an appropriate view).



Global visibility methods

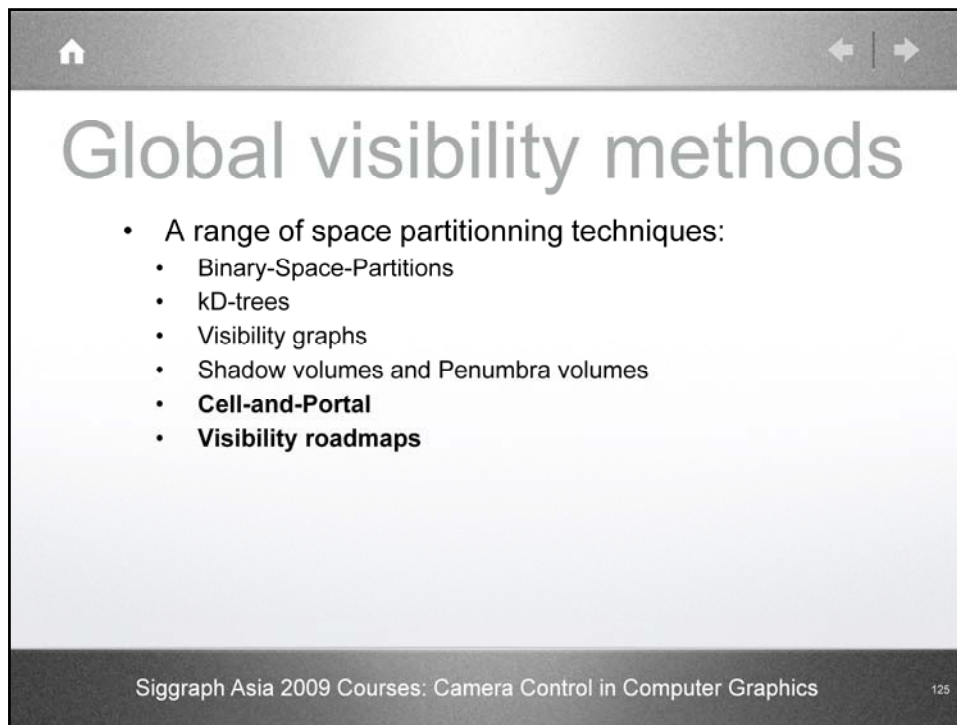
- provides a collection of techniques and structures to represent the global visibility in an environment:
 - centered on the notion of *visual events*





- separates the space into visible and non-visible areas
- two classes of problems
 - from a point: *from-point* visibility computation
 - from a region: *from-region* visibility computation
- techniques mostly pre-compute the visibility from the static parts of the environment

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics124

Visibility methods aim to calculate either the regions of a space which can be seen from a point (from-point visibility computation), or those that can be seen from a region (from-region visibility computation). In simple terms, visibility determination uses visual events - the boundary configurations for which the visibility changes - to partition space. Such methods can be broadly categorized according to the space in which the partitioning is performed, that is, object space, image space, viewpoint space or line-space (for a detailed presentation see [Dur99]). Visibility methods in dynamic environments have mostly addressed the problem of updating these visibility representations for moving objects [SG99] and modeling moving occludees (e.g. motion volumes [DDTP00]).

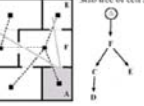
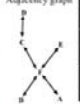

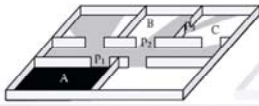


A large range of space partition techniques is present in the literature. In this course we'll present the cell-and portal techniques, and the visibility roadmaps.



Cell-and-portal Visibility(1)

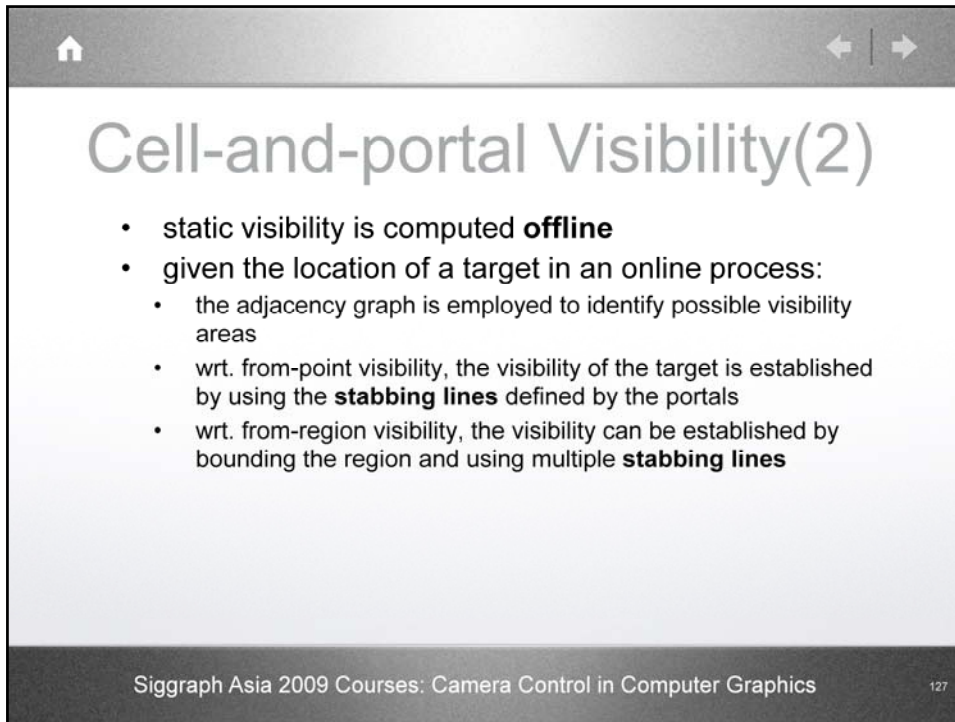
- based on architectural environments:
 - scenes are subdivided into rooms (the cells)
 - visibility occurs through openings (the portals)
 - the cells are connected in an adjacency graph
 - each cell visibility is then established by propagating visibility in the adjacency graph and checking the portal-to-portal visibility



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

126

C&P visibility is restricted to architectural environments, though abstract 2D $\frac{1}{2}$ representations can be used to handle more complex scenes [Lam09]. C&P techniques have been initially proposed to improve occlusion culling in complex urban scenes (ie removing parts of the geometry that are hidden). The scene is decomposed into cells (or convex cells to ensure full visibility inside them – a constrained Delaunay triangulation helps to compute such a decomposition), and cells are connected by portals (which edges are the support for visibility). Inter-cell visibility propagation is then performed by constructing stabbing lines (lines that separate the visibility in space). Visible cells are connected together in an visible adjacency graph.

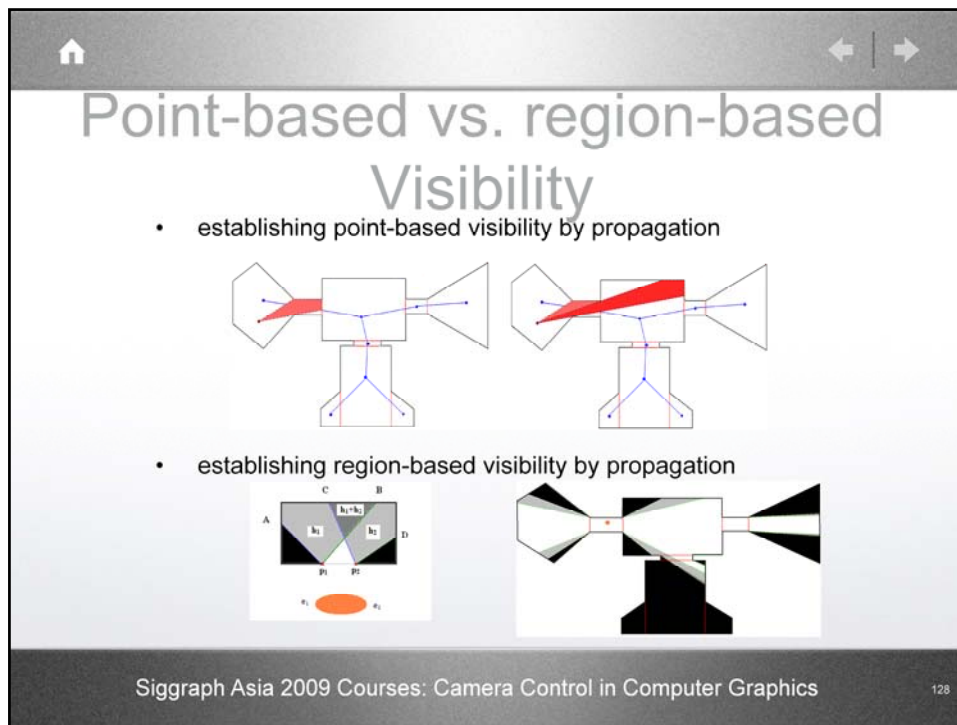


Cell-and-portal Visibility(2)

- static visibility is computed **offline**
- given the location of a target in an online process:
 - the adjacency graph is employed to identify possible visibility areas
 - wrt. from-point visibility, the visibility of the target is established by using the **stabbing lines** defined by the portals
 - wrt. from-region visibility, the visibility can be established by bounding the region and using multiple **stabbing lines**

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 127

Inter-cell visibility propagation is then performed by constructing stabbing lines (lines that separate the visibility in space). Visible cells are connected together in an visible adjacency graph.

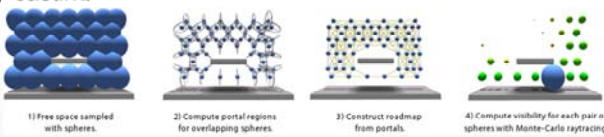


Examples of from-point visibility propagation (top view), and from-region visibility propagation (bottom view). The C&P decomposition helps to propagate the visible areas in cells.

Handling both static and dynamic elements

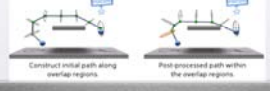
Coupling offline and online visibility [OSTG09]:

- first build a roadmap inside the environment (by sampling with spheres and building portals)
- second, compute offline visibility from sphere to sphere with ray-casting



1) Free space sampled with spheres. 2) Compute portal regions for overlapping spheres. 3) Construct roadmap from portals. 4) Compute visibility for each pair of spheres with Monte-Carlo raytracing.

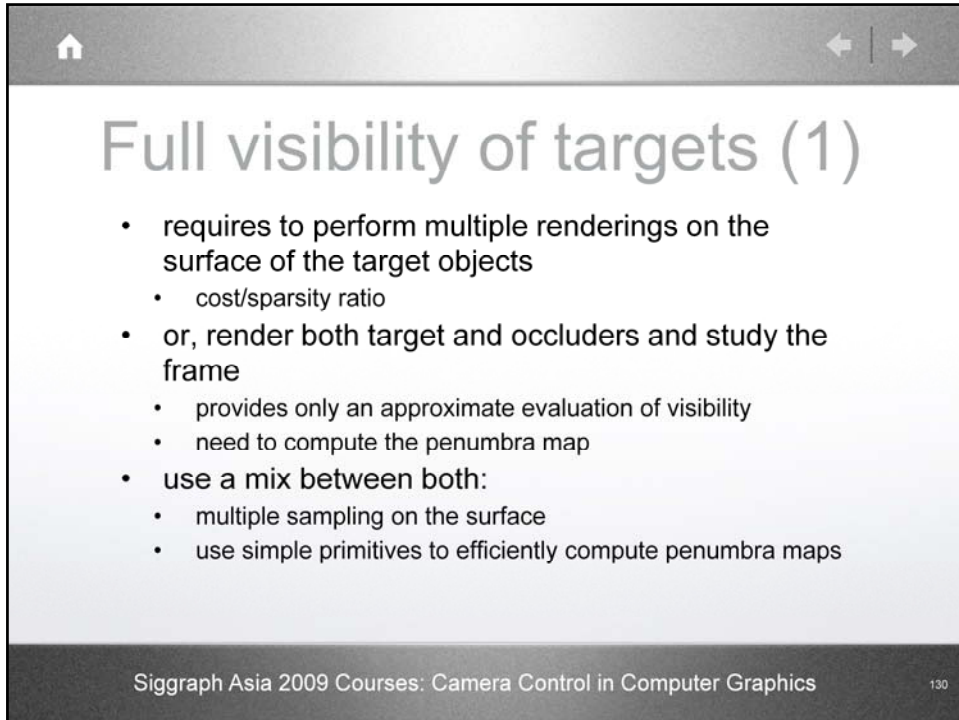
- third, use static visibility information to plan a path and use image-based techniques to handle dynamic occluders



Construct initial path along overlap regions. Post-processed path within the overlap regions.

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 129



Mandatory process to couple local visibility (for dynamic occluders) with global visibility.



Full visibility of targets (1)

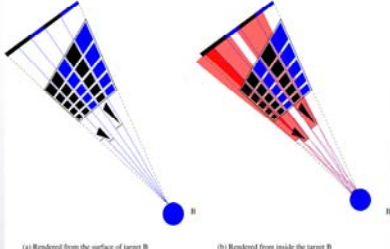
- requires to perform multiple renderings on the surface of the target objects
 - cost/sparsity ratio
- or, render both target and occluders and study the frame
 - provides only an approximate evaluation of visibility
 - need to compute the penumbra map
- use a mix between both:
 - multiple sampling on the surface
 - use simple primitives to efficiently compute penumbra maps

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 130



Full visibility of targets (2)



- From-region visibility computation:
 - locally approximate the surface of the target object as a rectangle parallel to the far rendering plane
 - perform a rendering that encloses the rectangle



(a) Rendered from the surface of target B (b) Rendered from inside the target B

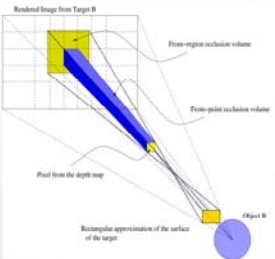
Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 131

From region visibility can be computed by using penumbra maps. The surface of the target object (B) is approximated by a set of patches considered as light sources. The negation of the cumulated penumbra areas provides a characterization of the target visibility.



Full visibility of targets (3)

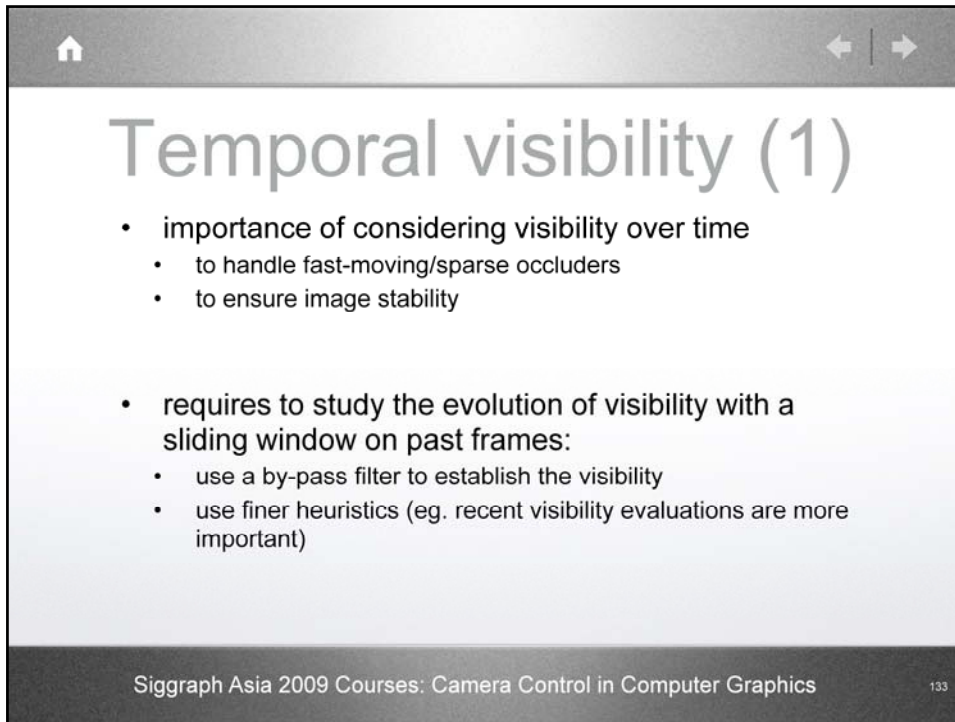
- perform a rendering that encloses the rectangle
- for each pixel of an occluder:
 - compute the penumbra map (considering the patch as the light source)
 - update the depth map accordingly



Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

132

The surface of the target object is decomposed or sampled with rectangular patches. Classic shadow map techniques are used in a first process. Then for each occluded depth pixel, we build the penumbra map by considering the patch as the light source. The depth map is updated with the penumbra information. Computations are strongly simplified by considering that the patch, depth pixel and depth map are parallel planes.



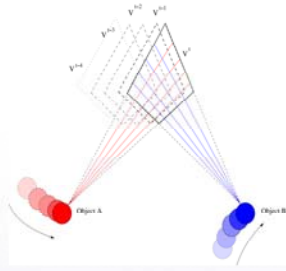
Temporal visibility (1)

- importance of considering visibility over time
 - to handle fast-moving/sparse occluders
 - to ensure image stability
- requires to study the evolution of visibility with a sliding window on past frames:
 - use a by-pass filter to establish the visibility
 - use finer heuristics (eg. recent visibility evaluations are more important)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 133

Another important problem for real-time camera control, and one that is related to visibility, is the maintenance of shot coherency (i.e. avoiding abrupt and visually incongruous changes in viewpoint). If a camera only reacts to features that are 'in shot', then the sudden onset of occlusion that occurs in dynamic and complex environments will result in equivalently abrupt responses by the camera. To mitigate this we need to incorporate mechanisms that stabilize the camera movements according to the spatial and temporal evolution of visibility. By monitoring the accumulation of the visibility information over the successive frames we can explicitly control the degree to which the camera is sensitive to partial and short-lived occlusions

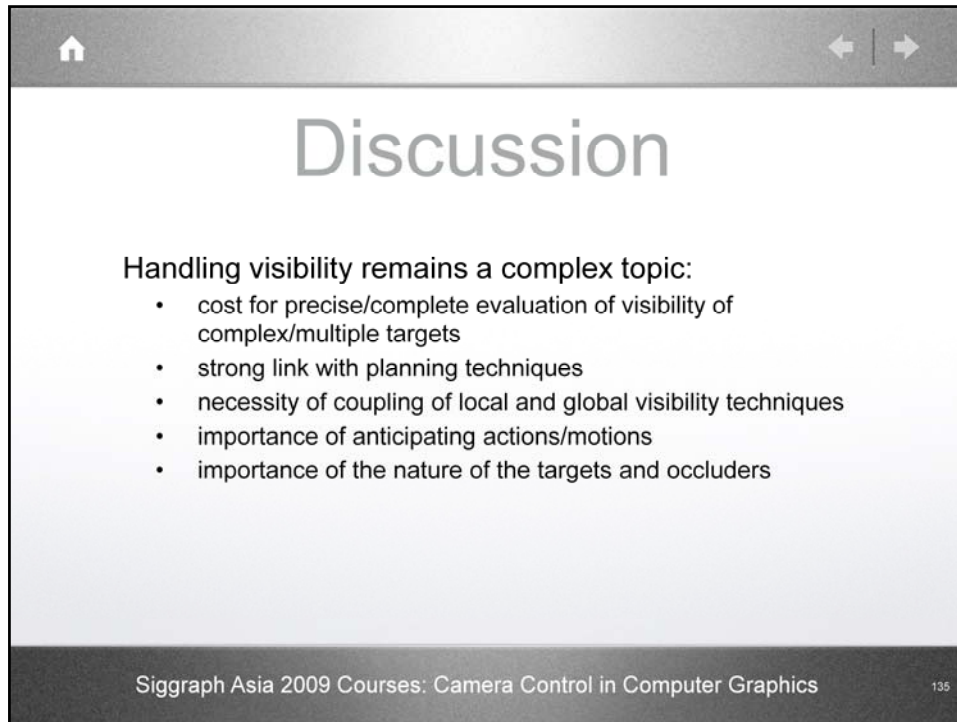
Temporal visibility (2)



- Accumulation introduces latency use a prediction model (eg. 10 frames ahead)
- and use a sliding window to only recompute the furthest locations (closest are considered precise enough due to scene coherency)

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 134

An adequate data structure can be easily designed to store a sliding window of shadow maps over time. The cost of accessing and cumulating visibility states is linear in the size of the sliding window.



⌂

⏪ | ⏩

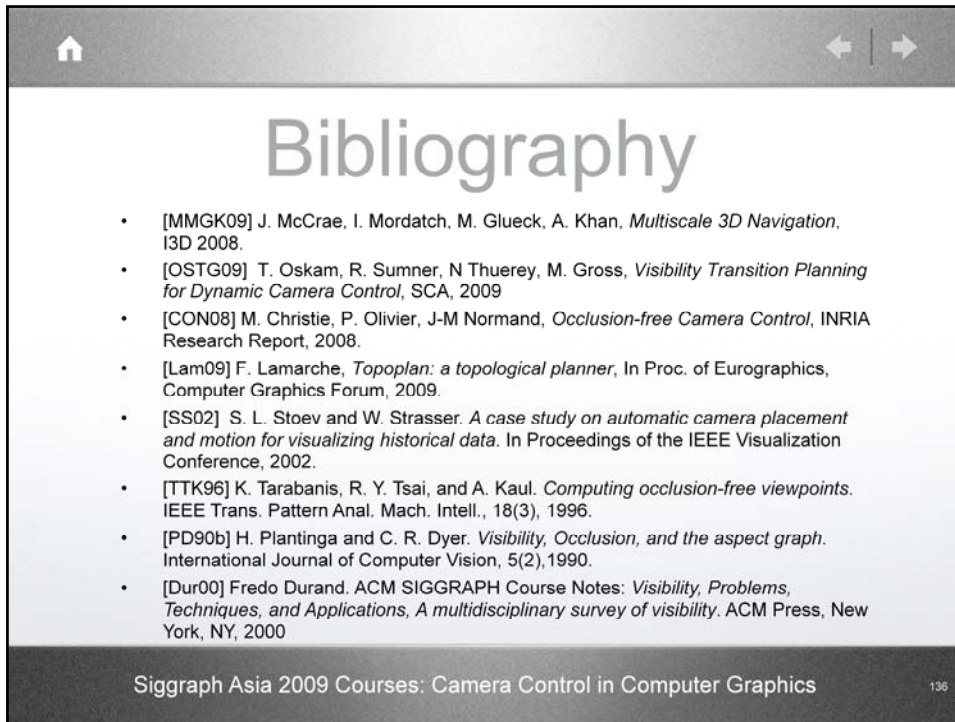
Discussion

Handling visibility remains a complex topic:

- cost for precise/complete evaluation of visibility of complex/multiple targets
- strong link with planning techniques
- necessity of coupling of local and global visibility techniques
- importance of anticipating actions/motions
- importance of the nature of the targets and occluders

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

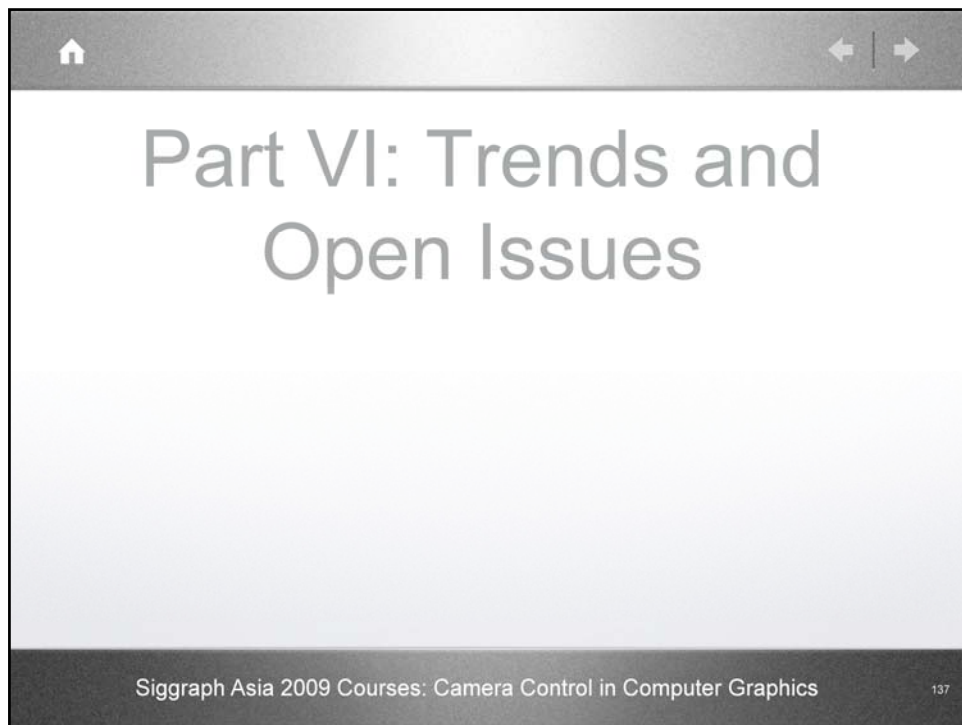
135



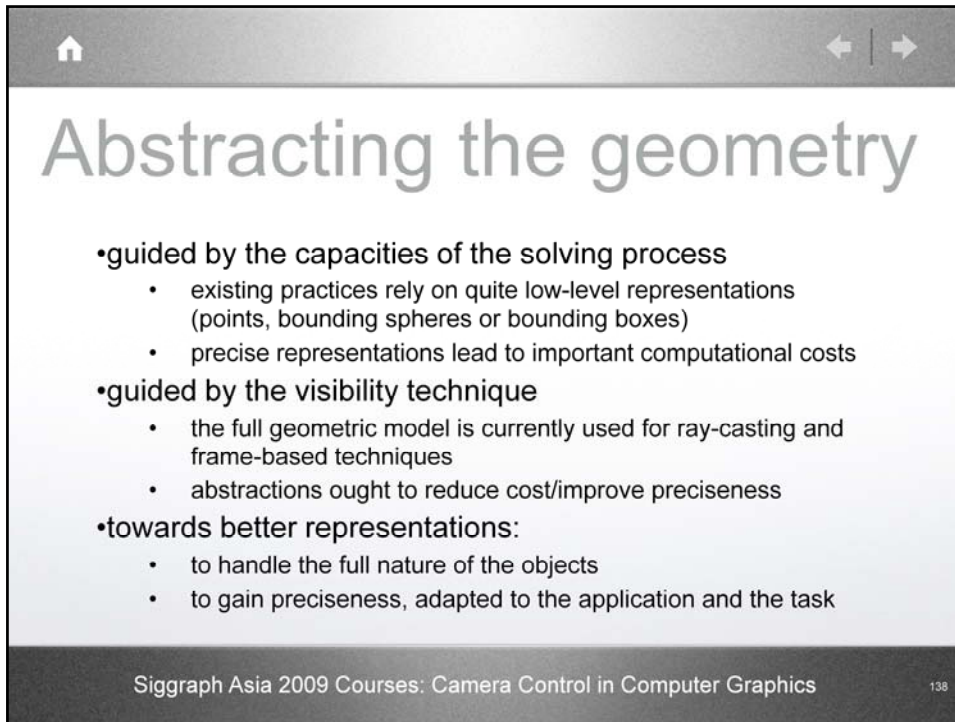
Bibliography

- [MMGK09] J. McCrae, I. Mordatch, M. Glueck, A. Khan, *Multiscale 3D Navigation*, I3D 2008.
- [OSTG09] T. Oskam, R. Sumner, N. Thuerey, M. Gross, *Visibility Transition Planning for Dynamic Camera Control*, SCA, 2009
- [CON08] M. Christie, P. Olivier, J-M Normand, *Occlusion-free Camera Control*, INRIA Research Report, 2008.
- [Lam09] F. Lamarche, *Topoplan: a topological planner*, In Proc. of Eurographics, Computer Graphics Forum, 2009.
- [SS02] S. L. Stoev and W. Strasser. *A case study on automatic camera placement and motion for visualizing historical data*. In Proceedings of the IEEE Visualization Conference, 2002.
- [TTK96] K. Tarabanis, R. Y. Tsai, and A. Kaul. *Computing occlusion-free viewpoints*. IEEE Trans. Pattern Anal. Mach. Intell., 18(3), 1996.
- [PD90b] H. Plantinga and C. R. Dyer. *Visibility, Occlusion, and the aspect graph*. International Journal of Computer Vision, 5(2), 1990.
- [Dur00] Fredo Durand. ACM SIGGRAPH Course Notes: *Visibility, Problems, Techniques, and Applications, A multidisciplinary survey of visibility*. ACM Press, New York, NY, 2000

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 136



This final session discusses the problems related to the actual deployment of these techniques and directions for future research, including (1) augmenting the expressiveness by considering cognitively well-founded perceptual and aesthetic properties; and (2) the development of editing constraints (for discontinuous shot camera transitions).

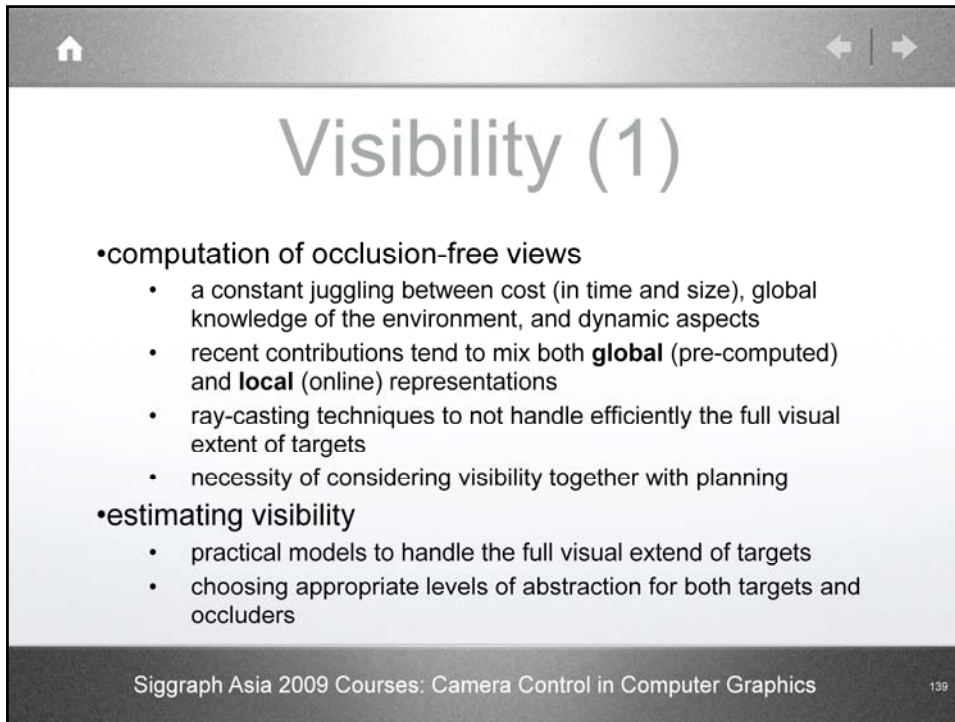


Abstracting the geometry

- guided by the capacities of the solving process
 - existing practices rely on quite low-level representations (points, bounding spheres or bounding boxes)
 - precise representations lead to important computational costs
- guided by the visibility technique
 - the full geometric model is currently used for ray-casting and frame-based techniques
 - abstractions ought to reduce cost/improve preciseness
- towards better representations:
 - to handle the full nature of the objects
 - to gain preciseness, adapted to the application and the task

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 138

The management of complex 3D scenes almost inevitably requires us to use abstractions of the underlying geometry. To this end, existing practice typically utilizes simple primitives: points or bounding volumes such as spheres. Some proposals do consider precise geometry for occlusion purposes [HHS01, Pic02], but at a significant computational cost. Improving the quality of abstraction of the objects is a difficult but necessary enterprise. The principal constraints on the abstraction to be used is the solving process itself and the computational resources available. In existing work abstractions are either based on geometric simplifications to reduce the computational cost (from points to single bounding and hierarchical regions), or on semantic abstractions (object, character) to aid description.

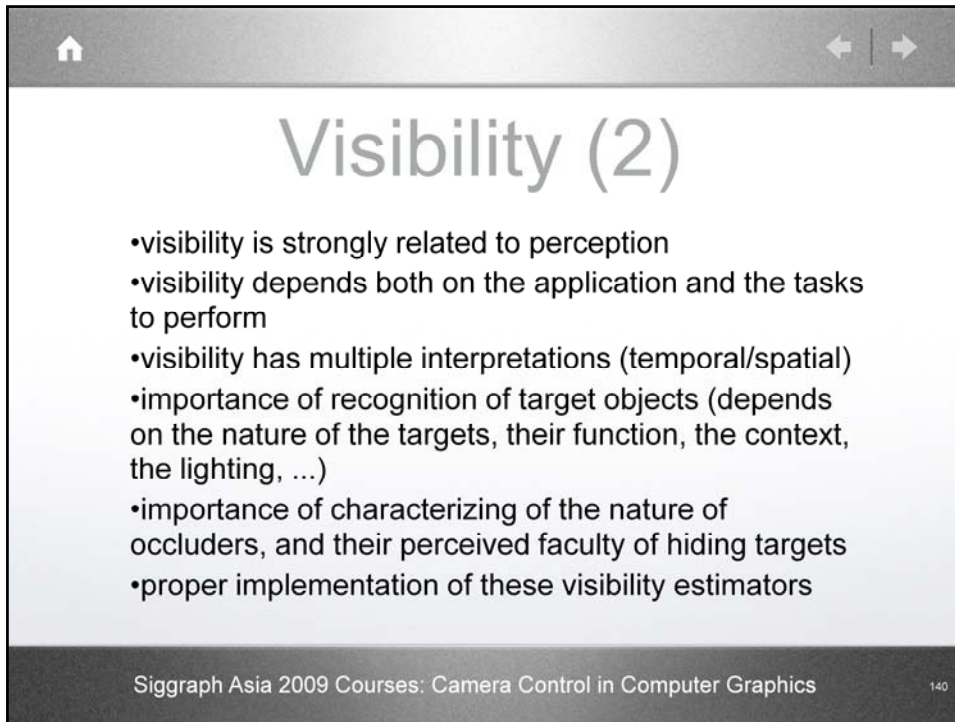


Visibility (1)

- computation of occlusion-free views
 - a constant juggling between cost (in time and size), global knowledge of the environment, and dynamic aspects
 - recent contributions tend to mix both **global** (pre-computed) and **local** (online) representations
 - ray-casting techniques to not handle efficiently the full visual extent of targets
 - necessity of considering visibility together with planning
- estimating visibility
 - practical models to handle the full visual extend of targets
 - choosing appropriate levels of abstraction for both targets and occluders

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 139

Visibility is actually one of the main issues in camera control, and has been quite neglected. With the advent of efficient dedicated graphical languages, such issues are currently re-explored.

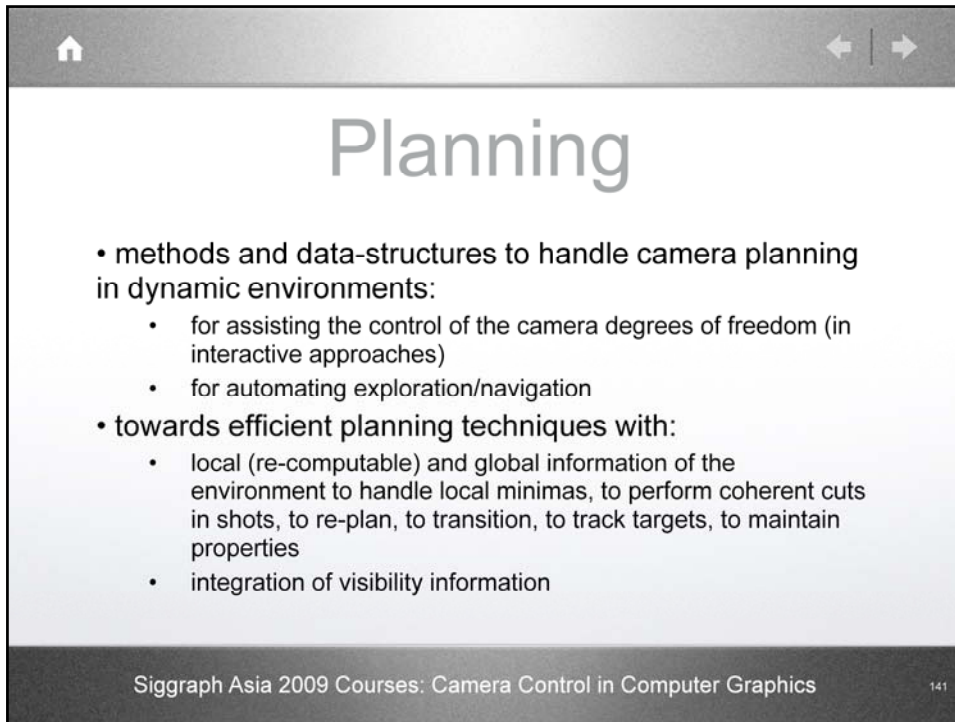


Visibility (2)

- visibility is strongly related to perception
- visibility depends both on the application and the tasks to perform
- visibility has multiple interpretations (temporal/spatial)
- importance of recognition of target objects (depends on the nature of the targets, their function, the context, the lighting, ...)
- importance of characterizing of the nature of occluders, and their perceived faculty of hiding targets
- proper implementation of these visibility estimators

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 140

Visibility is not only difficult from a technical point of view, it also is related to more fundamental aspects in perception that are critical to evaluate (recognizability, task-dependent, duration and extent of the occlusion).



⌂

◀ | ▶

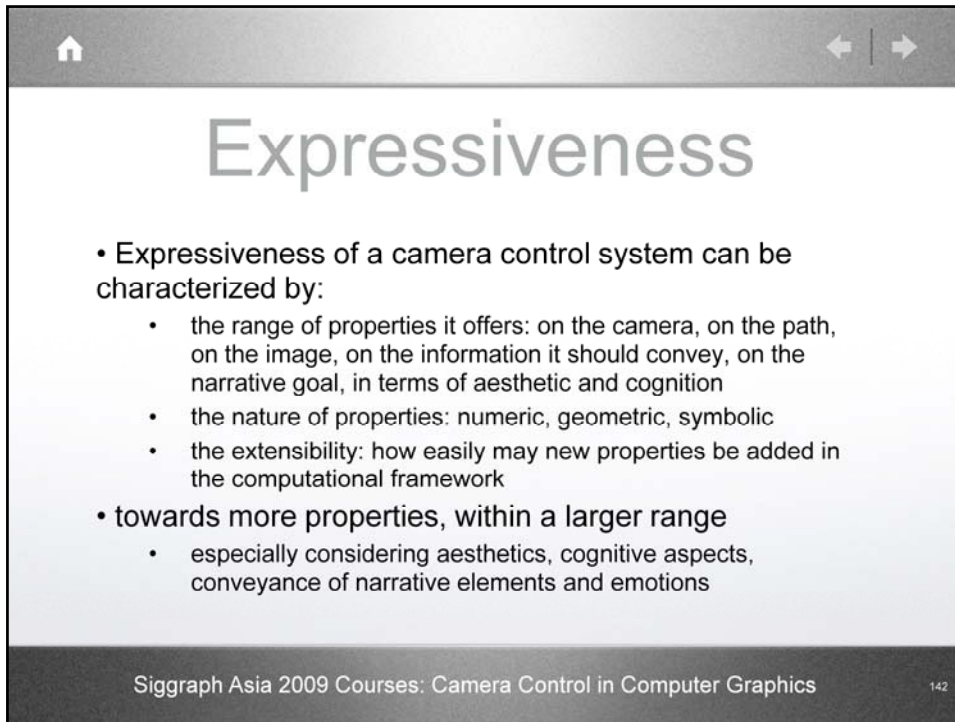
Planning

- methods and data-structures to handle camera planning in dynamic environments:
 - for assisting the control of the camera degrees of freedom (in interactive approaches)
 - for automating exploration/navigation
- towards efficient planning techniques with:
 - local (re-computable) and global information of the environment to handle local minimas, to perform coherent cuts in shots, to re-plan, to transition, to track targets, to maintain properties
 - integration of visibility information

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

141

Another major issue in virtual camera control. Though not all approaches rely on planning techniques (through-the-lens, reactive approaches), it remains central in navigation and exploration-based applications, either interactive or automated, and in games context.

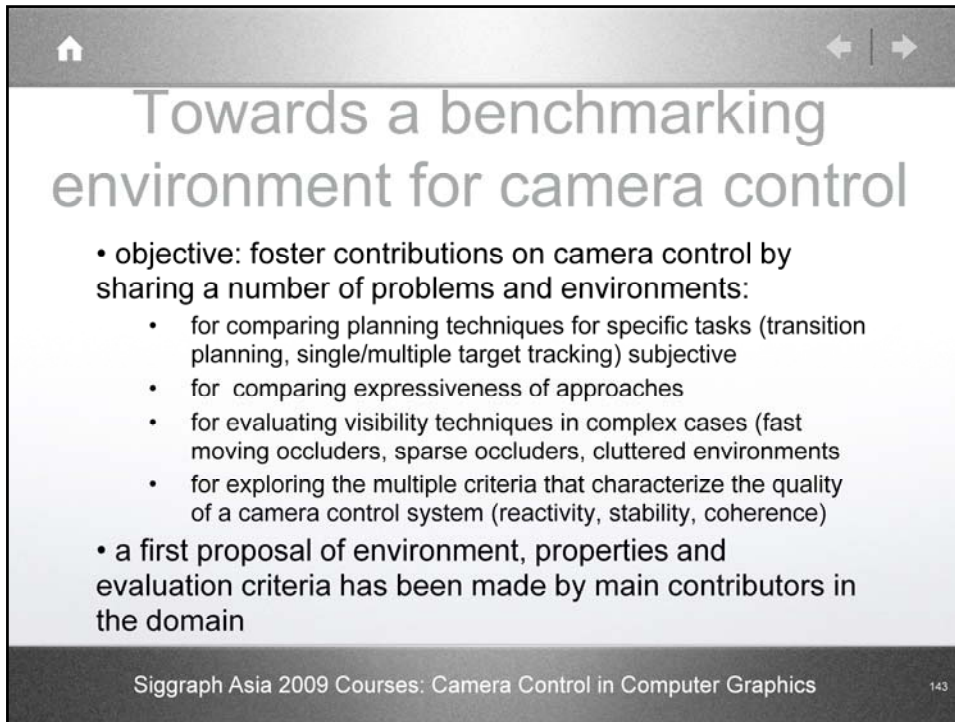


Expressiveness

- Expressiveness of a camera control system can be characterized by:
 - the range of properties it offers: on the camera, on the path, on the image, on the information it should convey, on the narrative goal, in terms of aesthetic and cognition
 - the nature of properties: numeric, geometric, symbolic
 - the extensibility: how easily may new properties be added in the computational framework
- towards more properties, within a larger range
 - especially considering aesthetics, cognitive aspects, conveyance of narrative elements and emotions

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 142

Expressiveness represents another major issue and a motivating research direction in virtual camera control. If we consider the 4 levels of information that an image or a sequence of images conveys (informative, perceptive, aesthetic and cognitive, and emotional), only the first level has been studied in detail (informative) and there are some significant contributions in the second (perceptive). Third and forth are pretty much in their infancy.



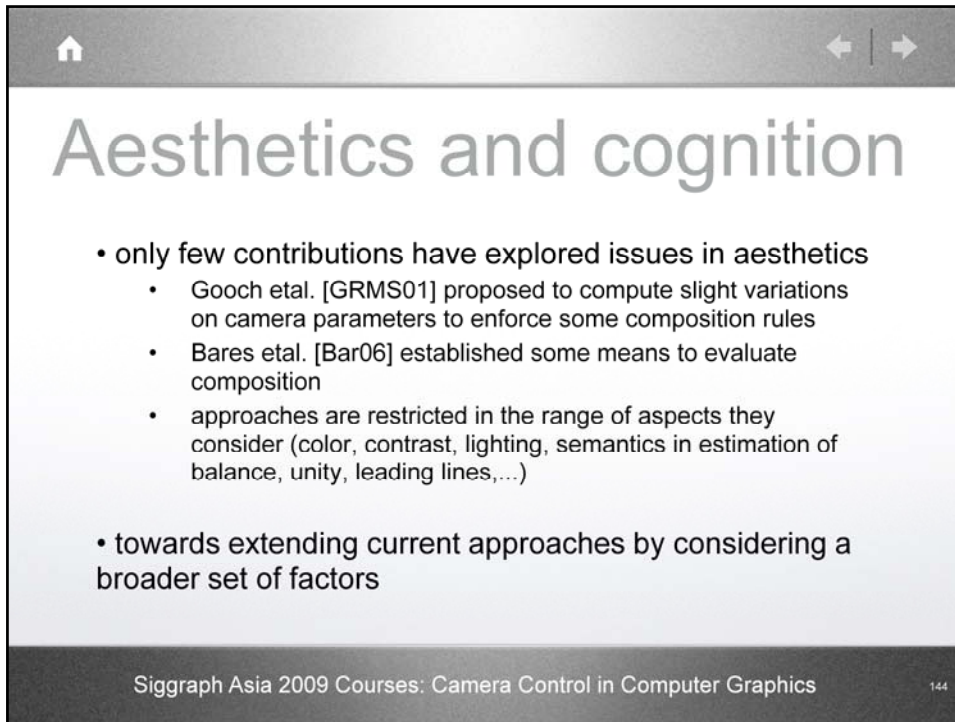
Towards a benchmarking environment for camera control

- objective: foster contributions on camera control by sharing a number of problems and environments:
 - for comparing planning techniques for specific tasks (transition planning, single/multiple target tracking) subjective
 - for comparing expressiveness of approaches
 - for evaluating visibility techniques in complex cases (fast moving occluders, sparse occluders, cluttered environments)
 - for exploring the multiple criteria that characterize the quality of a camera control system (reactivity, stability, coherence)
- a first proposal of environment, properties and evaluation criteria has been made by main contributors in the domain

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 143

A broad range of techniques have been proposed in the literature, that share a number of key issues. A proposal we believe in, is to create a benchmarking environment containing:

- a number of 3D example scenes
- for each scene, the description of a number of tasks to perform, with a uniform language description
- means and proposal of criteria to evaluate the quality of camera systems.

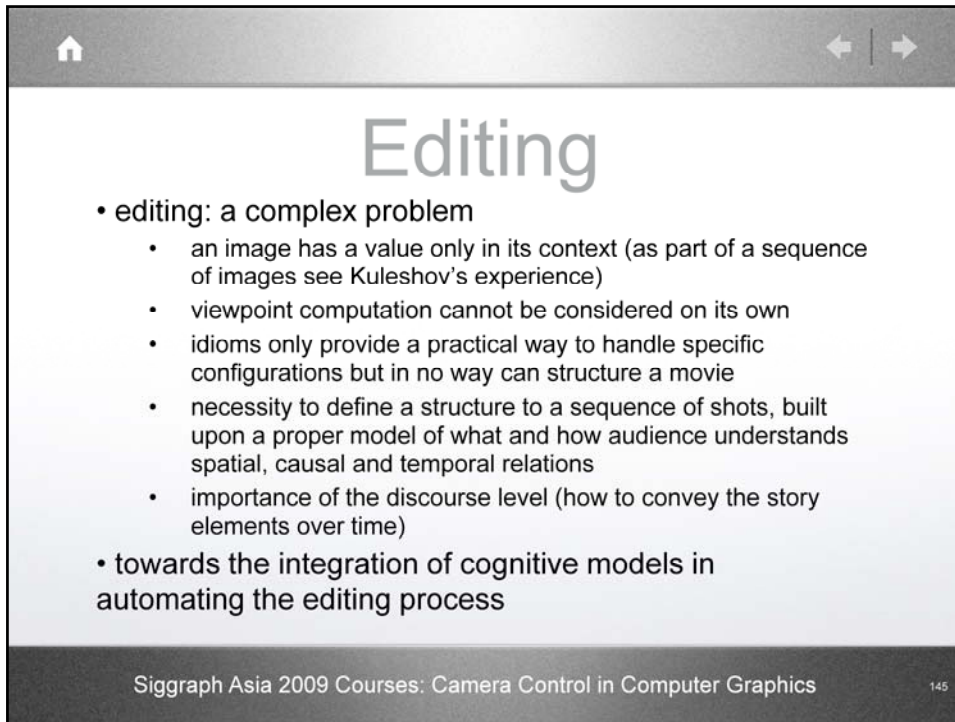


The slide is titled "Aesthetics and cognition" in a large, light gray font. It features a list of bullet points in a smaller, dark gray font. The slide has a dark gray header with a home icon and navigation arrows, and a dark gray footer with the course name and the number 144.

- only few contributions have explored issues in aesthetics
 - Gooch et al. [GRMS01] proposed to compute slight variations on camera parameters to enforce some composition rules
 - Bares et al. [Bar06] established some means to evaluate composition
 - approaches are restricted in the range of aspects they consider (color, contrast, lighting, semantics in estimation of balance, unity, leading lines,...)
- towards extending current approaches by considering a broader set of factors

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 144

A number of approaches characterize object recognizability and provide techniques to compute views that maximize the visual saliency for static scenes [HHO05]. Cognitive levels have been addressed both by the robotics and the computer graphics communities whereby a minimal set of viewpoints (canonical views) for recognizing an object and/or detecting its features is computed. The principal metrics are view likelihood, view stability [WW97], and view goodness [BS05a] which have been computed using entropy measures [VFSH01]. Extensions to automated navigation have also been explored for historical data visualization [SS02] and scene understanding [SP05], but remain the exception.



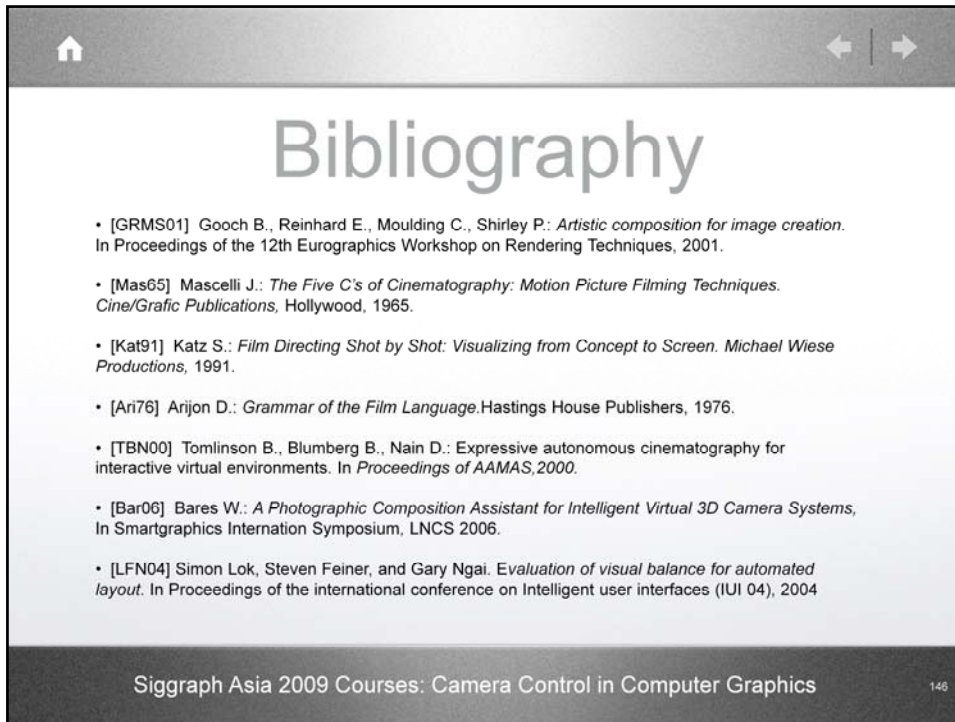
Editing

- editing: a complex problem
 - an image has a value only in its context (as part of a sequence of images see Kuleshov's experience)
 - viewpoint computation cannot be considered on its own
 - idioms only provide a practical way to handle specific configurations but in no way can structure a movie
 - necessity to define a structure to a sequence of shots, built upon a proper model of what and how audience understands spatial, causal and temporal relations
 - importance of the discourse level (how to convey the story elements over time)
- towards the integration of cognitive models in automating the editing process

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics 145

Whilst the aesthetic properties are likely to be founded on an adequate cognitive model, work on exploiting editing rules to effectively engage the user are

still very much in their infancy. We see this as a key area for interdisciplinary research by both computer scientists and cognitive psychologists.



Bibliography

- [GRMS01] Gooch B., Reinhard E., Moulding C., Shirley P.: *Artistic composition for image creation*. In Proceedings of the 12th Eurographics Workshop on Rendering Techniques, 2001.
- [Mas65] Mascelli J.: *The Five C's of Cinematography: Motion Picture Filming Techniques*. Cine/Grafic Publications, Hollywood, 1965.
- [Kat91] Katz S.: *Film Directing Shot by Shot: Visualizing from Concept to Screen*. Michael Wiese Productions, 1991.
- [Ari76] Arijon D.: *Grammar of the Film Language*. Hastings House Publishers, 1976.
- [TBN00] Tomlinson B., Blumberg B., Nain D.: Expressive autonomous cinematography for interactive virtual environments. In *Proceedings of AAMAS, 2000*.
- [Bar06] Bares W.: *A Photographic Composition Assistant for Intelligent Virtual 3D Camera Systems*. In Smartgraphics International Symposium, LNCS 2006.
- [LFN04] Simon Lok, Steven Feiner, and Gary Ngai. *Evaluation of visual balance for automated layout*. In Proceedings of the international conference on Intelligent user interfaces (IUI 04), 2004

Siggraph Asia 2009 Courses: Camera Control in Computer Graphics

146

Chapter 4

Reference Papers

4.1 Paper 1: [HHS01]

Nicolas Halper, Ralf Helbing and Thomas Strothotte, *A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence*. In Proceedings of the Eurographics Conference (EG 2001) (2001), vol. 20, Computer Graphics Forum, pp. 174-183.

A Camera Engine for Computer Games: Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence

Nicolas Halper Ralf Helbing Thomas Strothotte

Department for Simulation and Graphics, University of Otto-von-Guericke, Magdeburg, Germany

Abstract

Many computer games treat the user in the "1st person" and bind the camera to his or her view. More sophistication in a game can be achieved by enabling the camera to leave the users' viewpoint. This, however, requires new methods for automatic, dynamic camera control. In this paper we present methods and tools for such camera control. We emphasize guiding camera control by constraints; however, optimal constraint satisfaction tends to lead to the camera jumping around too much. Thus, we pay particular attention to a trade-off between constraint satisfaction and frame coherence. We present a new algorithm for dynamic consideration of the visibility of objects which are deemed to be important in a given game context.

1. Introduction

The current use of camera techniques in computer games is comparable to the situation in the early days of motion pictures. Back then, actors and directors were used to the theater stage and the viewer had only one perspective for the whole performance. There was no moving camera, no cuts, and only one shot size. Over the following decades, cinematography developed and people became accustomed to its language. Today, camera techniques form an important part of the story-telling process. The right use of a camera can enhance a viewer's experience as much as poor camera handling can destroy it.

In the past, games provided a first-person view or allowed the gamer to choose from one of several predefined camera positions including one or more over-the-shoulder views. The use of first-person views places the gamer inside his player character and has the fortunate side effect of freeing the game developer from any serious camera work. This is a valid approach for the type of first-person shooters that have dominated the real-time 3D game sector, but there is now a need for a more sophisticated camera handling. Indeed, the camera settings can reveal information to the user in a subtle way. Third person views, for instance, are often associated with higher player-character identification, so that the player takes on the role of his or her hero (or in today's market, more likely the heroine).

Games proudly present their optimised graphics engine, their superior artificial intelligence, even their sophisticated story engine. In this paper, we propose methods to create a further step in the evolution of interactive entertainment – the games camera engine. One of the main conceptual challenges is to obtain a balance between optimality of camera settings on the one hand, and smooth transitions with appropriate cuts on the other (i.e. frame coherence). We begin by focusing on previous work (Section 2) in this area, then outline the requirements for a camera engine, so that we present our system in Section 4. Finally, we apply the camera engine to some scenarios for evaluation, before we conclude in Section 6.

2. Previous Work

Much work has been done on enabling users to directly manipulate the camera^{14, 20, 25, 24} or on providing camera assistants^{22, 12, 16}. However, relatively little has been done in immersing the user in an environment by controlling an object or character, whereby the camera process should be invisible to the user. Nevertheless, allowing communication of important visual goals such as navigational information, or notable features in the environment, should be enabled so that the player is not left running into the unknown.

2.1. Background

We agree with the game developer Barwood³, who notes that

"[computer games] are as different from [movies]
as movies are different from theater"

This means that camera techniques used in future games will have to develop and employ languages and rules on top of cinematographic techniques. Differences arise from the fact that computer games are highly interactive (unlike film or theater). This makes it impossible for any director module to stage and rehearse actions beforehand. Therefore, even if computer games' camera modules would possess the cinematic knowledge of directors, they could not apply it since these techniques depend to a great part on trial and error. Also, when shooting a real movie, directors are at liberty to reposition actors and change the scene, even the script. In contrast, a camera module is typically given the scene as it is. If, due to geometric constraints, a good camera position can not be found in the scene as it evolved up to that moment, a less-than-satisfactory camera position must be used. Thus, converting cinematographic techniques into idioms, such as He et al.¹⁵, will provide opportunities for predefined film scenerios, but lack camera setups to convey visual goals that aid interactivity for the player.

The use of the camera in a movie is highly dependent on the current situation. To approach the quality of film's camera techniques in computer games, games can classify each possible situation during the game into a (hopefully) small number of variants. Each of these can be associated with a number of techniques describing how to shoot that scene, where to place virtual cameras, how to follow an actor and when to cut (switch between virtual cameras). For instance, in a dialog situation, the camera can focus on the participants alternately as they speak, starting with an establishing shot framing both speakers. When exploring unknown territory, the camera – which normally follows the player character – can provide hints for the gamer by looking elsewhere if there are important clues to be found (or missed). In interactive applications, all shots are presented to the viewer immediately without any editing and montage of the raw footage, therefore a director engine should also know what is likely to happen next so that it can plan shots and their transitions (panning or cuts) in advance.

2.2. Computer Games

Game companies such as Lucasarts have a great series of adventure games in 3D, but have rigid fixed camera positions, so that if not everything is going to plan or not all characters are in place, it will not be shot right. Therefore they must limit their interactions based on pre-defined scenarios.

Most real-time games solve camera positions procedurally, using specialised camera routines adapted to the design of each level. This makes for an inflexible camera engine,

and often leads to situations where the camera is not showing the best view for the user. In Tomb Raider, for example, the camera can produce awkward views in closed spaces when Lara (the heroine) is backed up against a wall, because the camera computes without explicit consideration for visual properties in the view.

2.3. Constraint Satisfaction

Few interactive systems propose methods that effectively avoid occlusion. This is surprising considering the importance of maintaining an objects' visibility in shot. Christianson⁶ check for occlusion given a camera shot by testing against overlapping bounding spheres and incrementing an occlusion counter, but do not adjust camera state to accommodate visibility. Work done by Halper and Olivier¹³ use image precision calculations to assess visibility, but employ offline genetic algorithms for optimal camera state generation. Tomlinson²³ test against occlusion in their interactive environment by shooting a ray to the object in question, and adjust the camera vertically only, although this method is mostly suited for use in sparse outdoor environments.

Philips²² developed an algorithm based on the hemicube⁷ to select viewpoints and viewangles to ensure that a manipulated object is not obstructed by other objects. They use a cube centered around the origin of the object being manipulated, and orient it towards the camera position, whilst projecting the geometric environment onto each cube face to achieve a visibility map. If the camera is obstructed, they look in the neighborhood of the direction of the camera for an empty area in the visibility map, which suggests a location of the camera from which the object will be visible. Drucker⁸ processes the visibility map to create a potential map, which is followed from the initial location down to the nearest location in the visible region. Philips²² acknowledges that their algorithm will often fail in an enclosed environment without the use of depth information in the visibility map. Bares¹ accomodate this fact when their optimal vantage angle is occluded by decreasing the distance of the camera to the object until it is placed in front of the nearest obstacle. However, none of these techniques can compute a position such as P shown in Figure 1, and can only resolve visibility for a single point. Our algorithm described in section 4.4 is able to find points such as P, and resolves occlusion constraints for an arbitrary number of points.

Drucker et al.^{9, 10, 11} set up an optimal camera position for individual shots subject to constraints. The camera parameters are automatically tuned for a given shot based on a general-purpose continuous optimisation paradigm. These methods proved effective and allowed the design of camera modules. However, the camera modules, such as used in the CINEMA system, experience problems combining and constraining multiple procedures, requiring specifically tailored procedures to be setup. Bares¹ are limited to using vantage angle, viewing distance, and occlusion avoidance.

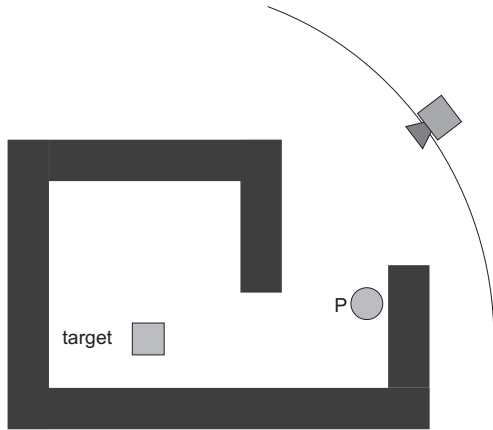


Figure 1: Figure showing problems when limited depth information is available in the visibility map. Current visibility algorithms do not process depth information across other angles of visibility, such that a viable point P can be returned

Neither provide predictive analysis for their interactive environments, or offer the ability to impose constraints based on existing camera situation and movement. This in turn means that they cannot achieve a high level of camera frame-coherence.

2.4. Cinematography Concepts

Christianson⁶ claim that shots are guaranteed to result in a common shot form using their Virtual Cinematographer, albeit with a greatly reduced set of possible camera specifications.

Systems such as CATHI⁵ or by Karp and Feiner^{17, 18} encode idioms in the form of film grammars, using a top-down approach in generating a sequence of shots. They are able to produce camera paths that achieve certain visual goals, but use timing information in the animation and therefore cannot be directly applied to reactive environments like computer games.

Tomlinson²³ propose a behavior-based autonomous cinematography system by encoding the camera as a creature with motivational desires, focusing on camera movement styles and lighting in order to augment emotional content. This is a step in the right direction for autonomous camera agents in interactive worlds, as their camera creature attempts to capture sequences that are of interest to the viewer. However, their constraint solver is computed procedurally based upon requirements for their system, losing the flexibility desired for developer-defined specifications.

3. Camera Engine Requirements

We propose that a camera module should be a part of the game engine pipeline as shown in Figure 2. Each module

generates its own output down the pipeline from the given inputs. The story engine drives the motivations for the actions in the game, and is the most flexible and creative part of the project. The action module creates events – interactions from the player with the environment and story-related actions. On the bottom extreme of the pipeline, we find the renderer. This engine must produce consistent crisply defined results. The lighting module is a step higher, but is dependent on adjusting settings to emphasize visual goals depending on the camera state. The camera module finds itself in the middle of the pipeline, a balance between flexibility and hard constraints – it must try to convey visual goals that dramatize the action, but is still confined to properties of the environment.

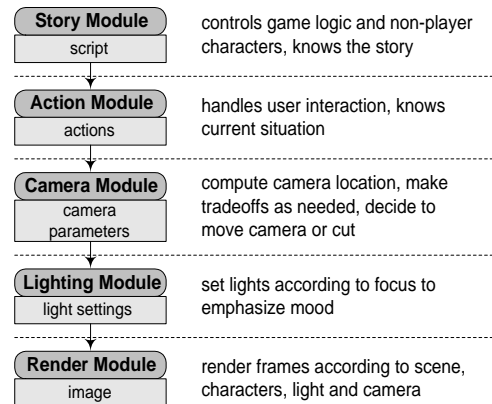


Figure 2: The game pipeline

On the basis of our analysis of previous work, we can reduce the camera engine requirements down to three basic requirements:

flexibility: must use parameterised techniques and versatility in defining constraint specifications in order to adapt to the output of events from the action module.

information: the more the camera knows about what is going on in the world, the better. We need event calls, information from actors, player motivations, and visual goals.

satisfaction: a best-fit solution will not always be present. Therefore, we need partial satisfaction solutions, and incorporate adaptive degradation, so that we can at least convey the most important visual cues at any given time.

3.1. Specification

A director does not plan shots by their low-level parameters such as camera position and direction, but by their desired visual properties, as first suggested by Blinn⁴ and applied by Drucker¹⁰. These can be put as constraints on the

camera but need not immediately lead to concrete parameters (which can be influenced by emotion templates). An automated camera control system must therefore find camera configurations that satisfy the constraints reasonably well. The first task therefore is finding these constraints.

Experience with previous work²¹ proved enough flexibility and expressive power is possible by constraint specification and made a first attempt at defining a list of shot properties to ensure a complete set of workable properties were available. This set was refined by Halper¹³, and Bares², producing similar results. The class of declarative constraints are refined here – we remove constraints that are either redundant or not useful in the context of computer games. We also find way to integrate these constraints into a constraint-solver pipeline detailed in section 4.3, that effectively integrates and satisfies specified combinations of constraints for arbitrary numbers of objects.

Level At The camera should be offset at a certain height relative to the object. This constraint often applies to portraits where the camera is facing the actor, or when we want to follow a target.

Angle to line-of-interest The angle from which to look at the target, specified relative to the line-of-interest, that is usually the line of interaction between characters or directly defined. This constraint is mutually exclusive with the next one.

Facing Each object has a vector that defines the ‘front’ of the object. Since this direction is tied to the target, the camera moves when the target turns. From this we can also specify informative 3/4 viewing angles to objects, or create over-the-shoulder shots by setting the desired viewing angle to look from behind the object.

Size This is actually used to control the camera’s distance to the target. Since finding the right distance depends on the targets shape and size as well as viewing angle and focal distance, the camera distance is better specified in terms of the resulting size of the targets projection on screen. When maintaining constraints over time, blind adherence to a size constraint dependent on camera or object orientation can result in oscillating camera movements as the size measures vary (e.g. this would occur if we measure size as the relative number of filled pixels in the view for an object). Therefore, direction invariant metrics such as the bounding sphere radius are a better choice than constraining against more precise measures.

Height angle Instructs the camera to watch the target at a specified angle from above or below.

View at angle (X and Y) Position the camera so that the line from target to camera has a specified angle to the viewing direction. If set at 0.0, this puts the target in the center

of the screen. X angles other than 0.0 move the projection of the target to the left or right side of the screen; the Y angle is used to push the target toward the top or the bottom. For the target to remain on screen, these angles should be set smaller than the camera’s field of view in X or Y. We define position of objects as angles, so that it is possible to specify placements of objects surrounding the camera.

Visibility The target is to remain visible to the viewer, such that unwanted obstructions from scene geometry between the camera and target are avoided.

The constraints and specifications of this chapter have been designed so that they can sufficiently define a camera position and viewing direction. Some combinations of constraints will be impossible to satisfy, especially when constraints are specified across multiple objects at the same time. Therefore, if shooting more than one target, the sets of constraints must be carefully chosen to guarantee a solution. However, the constraint-solver may still try to find a partial best-fit for all constraints, as outlined in section 4.3.

4. Camera System

We are now in a position to design a system which incorporates the concepts introduced in the last chapter. The system diagram is shown in Figure 3, and the various components that comprise the system are described below.

4.1. Trade-Offs Between Constraint Satisfaction and Frame Coherence

Note that constraints are based on targeted objects and have to be re-evaluated as they move. In addition, events generated by the story and action modules can change each frame, producing different visual goals and constraint specifications for each situation. A purely reactive application of constraints, such as Bares¹, will give rise to “jumpiness” as the camera constantly jumps to global best-fit solution spaces, in particular when avoiding viewing obstructions. To address jumpiness we have to maintain frame-coherence. Since frame-coherence makes for smooth camera motion, it must be given priority over strict constraint adherence. We address this problem by using several factors: section 4.3 constructs a constraint-solver pipeline that computes new positions from existing camera state, Section 4.3.1 adds relaxation parameters to the constraints, whereas section 4.4 details the occlusion avoidance algorithm. Finally, Section 4.5 uses lookahead algorithms to adjust the camera to future situations.

4.2. Director

The director takes input from the action list generated from the previous stage in the game pipeline. The Predictive Camera Planner requests settings for a certain time t . Based on

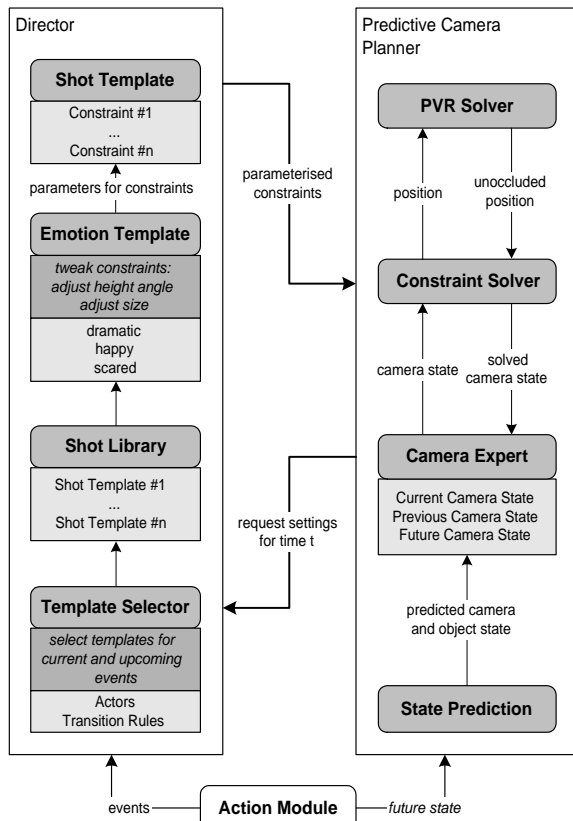


Figure 3: The overall design of our camera system

input from the event list, the director selects templates using the Template Selector, and prepares the constraints for the Constraint Solver. The director selects templates available from the shot library, using a number of transition rules so that successive shots *fit* together.

Emotion Templates encode various factors that may influence the results of camera shots. For instance, shot specifications can show a certain style of shot, whereas dynamic modifications to those specifications can produce various moods. We can influence the parameterisations of the shot properties defining a shot such that they accentuate a certain effect. For instance, we may add a height angle to the camera, and produce a "moody" effect. The amount of "moodiness" can influence the scale of the alteration to the height angle.

4.3. Constraint Solver

To achieve frame-coherence we compute new solutions for camera state based on existing camera state. Thus, we plan the position of the camera for the next subsequent frame. Since not all constraints can be fully satisfied for each change in the scene, we get approximate results by first solving for certain constraints, and then modifying the camera

state to accommodate the other specified constraints. The end result is one that best approximates the desired output, and allows a variety of heterogeneous constraints to be integrated and put to practice.

Figure 4 shows how these constraints are applied. Note that each successive constraint in the constraint-solver pipeline minimally influences the previous adjustments. Figure 5 shows what happens when constraints of the same type are used for multiple objects. There we test for cases and adapt to multiple settings and reach approximate results. Alternatively, we get more precise results for specific visual goals by using a special constraint combiner, that explicitly has an algorithm to solve. For instance, we are able to solve for a size and viewport position constraint on one object, plus an additional position in viewport constraint on another.

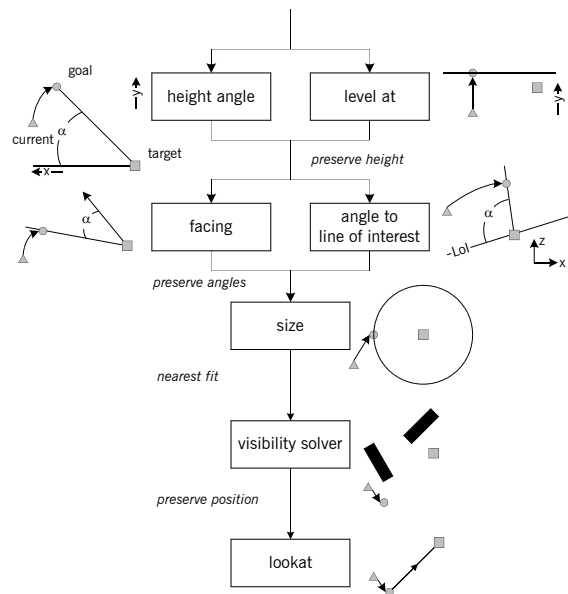


Figure 4: The order in which constraints are applied. The current camera state is shown as a triangle, the target as a square and the goal state as a circle. Each successive constraint minimally influences output of previous constraints. For instance, size conserves both the xy and xz -angles created from (height angle/level at) and (facing/angle to line-of-interest).

4.3.1. Relaxing constraints given tolerance settings

All constraints cannot be instantly satisfied for each change in the scene, since this produces a rigid camera feel. Therefore, each constraint has an optimal setting (e.g. size 30%) which defines a goal for the camera. A tolerance region is also specified, so that if the camera does not lie in the optimal state, we can compute how far it is from this goal. Depending on relaxation parameters, the camera can be placed a ratio p closer to this optimal. Applied to each frame in the

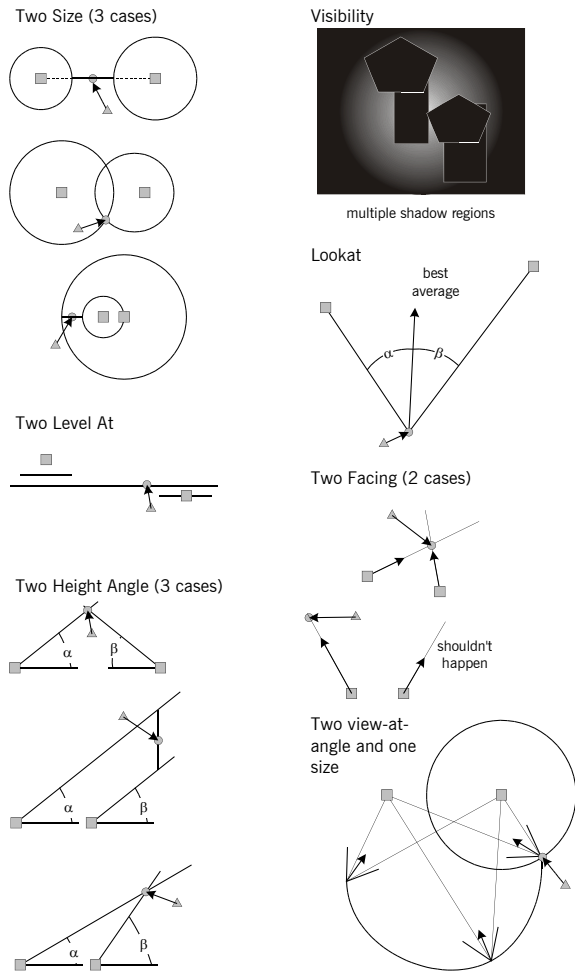


Figure 5: Cases of using 2 targets per constraint. Many of these can be adapted to using 3 or more targets.

sequence, this acts in similar fashion to a local optimization search, as the camera fluidly moves to a more satisfiable region. In the case that the camera lies outside the tolerance region for a constraint, then it must be placed at the borders of the tolerance region. If too many constraints are outside their tolerance regions, then the camera may select an additional set of visual properties for the shot, or use a transitional cut.

4.4. Potential Visibility Regions

The technique we use to compute a new camera position that provides an unobstructed view of points of interest addresses the shortcomings of the algorithms outlined in Section 2.3. A flexible and robust method is introduced that allows user-definable visibility goals to be applied to an arbitrary number of points.

We allow constraints on the camera in the processing of

occlusion avoidance by using what we call Potential Visibility Regions (PVR). The developer/designer can impose constraints on camera movement for the task of occlusion avoidance by defining a set of geometric constraints using polygons. To denote preference, polygons are shaded a brighter color than those geometric regions of less preference. This geometry defines the PVR to which the camera may move in order to obtain an unobstructed view of the points of interest.

In order to find the best position that satisfies visibility requirements, we write only depth information from the potential occluders to a visibility map. Then, we render the potential visibility geometry to the buffer in order of most desirable regions (with the brightest colors) to least desirable regions (darker colors), stencilling the regions which were rendered first. The result is an image buffer, whereby the brightest colors that first pass the depth test are visible. The brightest color in the image buffer, therefore, denotes the most desirable position for the camera to move to.

The flexibility in this algorithm is in design of the PVR. As an example, Figure 6 demonstrates how we would draw geometry in order to find the closest distance from the camera to an unoccluded view. In addition, we can constrain the camera such that it may move closer to the object or along the world-coordinate y-axis only. The choice of the PVR can be dependent on the tolerance settings of other constraints. For example, if we have a hard constraint on viewing angles, we would use the rightmost example shown in Figure 6. If we had a hard constraint on size, we could draw a bounding sphere around the object of interest with radius set to the distance defined by the size constraint, such that visibility solutions will only be found on this sphere, maintaining full satisfaction of the size constraint. These visibility regions also have the possibility to be defined without modifications to the core algorithm – they can be input as geometric data at run-time.

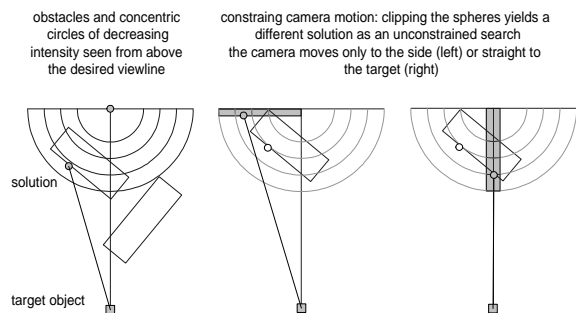


Figure 6: Demonstrating sample use of rendering camera solution regions, selection is based first on highest color, then on closest angle. Using solution regions other than spheres allows camera motion to be constrained further.

The introduction of PVR defined by geometry, combined

with the advent of new shadow casting algorithms and hardware, allows us to extend the visibility search for multiple points of interest, such that we are able to find a location for an unobstructed view for each point of interest in real-time.

A projective shadow casting algorithm²⁶ can be rendered in real-time with today's graphics chips, such as through the use of nVidia's register combiners. Alternatively, this may be done by stencilling shadow regions generated by object silhouettes¹⁹. In either case, we are able to process shadows upon our PVR from each point of interest, such that our final result is a cumulative shadow from all our points of interest. An appropriate view in which to render the potential visibility geometry is computed (we take the point closest to all points of interest). From this, we render only the potential visibility geometry that is not shadowed by any occluders, so that only the locations where each point of interest is visible comes through to the final image map (see example in figure 7).

The cost of solving visibility using multiple points is linear wrt. the number of points we solve for. For each point, we need to render a view (depth-write only) and read from a depth buffer. This buffer can be reduced to as little as 32x32 pixels to reduce polygon-fill and read-buffer overhead, and gives approximative results that may produce inaeesthetic shadows for visualisations purposes, but serve well for occlusion information. Note also, that the occluding geometry need not be as complex as the actual visualised geometry by the player, allowing the further reduction of rendering cost by using coarser occluder geometry as a representation of the actual finer detailed models.

4.5. Predictive Camera Planning

A camera cannot make an intelligent move without at least considering a future situation.

In order to achieve higher frame-coherence and smoother camera movement in a reactive environment, the camera is to progress consistently from frame-to-frame, without disorienting the player with rapid swinging camera movements. We are able to make some guess as to where objects and the camera are likely to be a given time t in the future. To do this, we use approximative calculations for future scene and object state based on past trajectory and acceleration information, and solve the camera for that predicted state. The current camera trajectory is adapted so that the camera will be at this predicted position at the given time t , and we compute an estimated camera position for the next frame along this path. Now we apply the constraint solver on that expected position, to give the solved camera state for the next frame (see figure 8). Slight deviations to the expected may occur, but due to the fact that we recompute every frame, the camera is able to intelligently adjust ahead of time to a large number of situations.

If the increased cost of this process is too great, we may

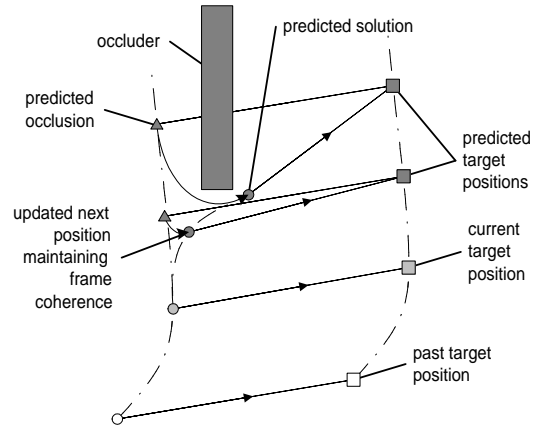


Figure 8: Altering the camera path to adapt to a predicted solved camera state.

alternatively *warp* the camera solution region geometry to accommodate camera motion characteristics, thereby implicitly encoding preference for the next best camera position along the current trajectory. An example of this is seen in figure 9.

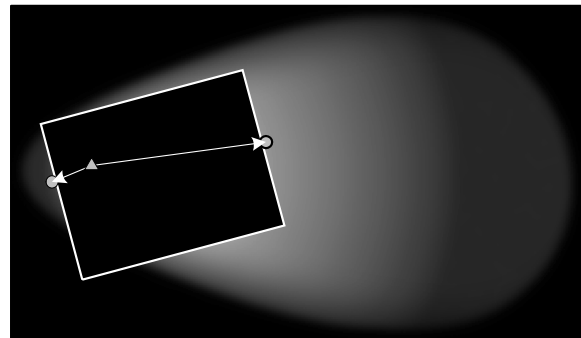


Figure 9: The camera solution region is warped according to motion characteristics of the camera. In this case, selected regions are favored in the direction of motion, and the algorithm returns the point to the right even though its angle of change is greater than that of the point to the left.

More effective techniques are also integrated, specific to the game context. One technique we use is to look along the viewing direction of the player. If the player can see into the distance, the camera adjusts to the player orientation to look in the same general direction. This gives the effect that when the player is moving and turning frantically in a corridor, the camera will only swing to give a view of the player heading when, for instance, the player turns to look through a door or window.

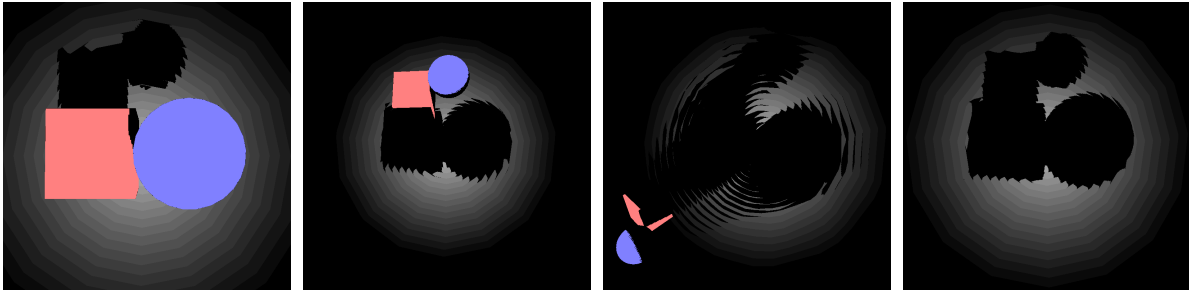


Figure 7: Multiple point visibility algorithm. The first two images show the view from two points of interest to the PVR, that define the shadow maps to be cast on the PVR using depth information. The third image shows an oblique view of the PVR and its occluder objects, showing that only those portions of the PVR that are unshadowed are rendered. The rightmost image shows the visibility map view of the PVR, from which we select the preference (brightest) color, and reverse-project from the z-buffer information to give the closest unoccluded view to which the camera should move.

5. Implementation and Evaluation

The more successful the camera engine is, the more subtle its satisfaction of visual goals will be. For this reason, we find it best to evaluate camera movement over specific scenarios. Demonstrations and videos of these results can be viewed on our website: <http://www.isg.cs.uni-magdeburg.de/~nick/cameraEngine>.

5.1. Exploration

To allow the player to explore her environment, we constrain the camera to focus on the object using a *levelAt*, *size*, *visibility* and a *lookAt* constraint. This gives the effect that the camera follows the player at player height, keeping her in unobstructed view at all times.

We have tested the exploration template on three different scenarios: 1) A helicopter flying through a city, 2) A human figure exploring inside a medieval building, and 3) a bee flying through a highly cluttered attic. The camera performs remarkably well in all three cases, avoiding obstructions and collisions, being capable of adjusting appropriately to many situations, despite the varied spatial arrangements and complexity of each environment.

5.2. Effects of Predicting Camera and Environment State

Here we show and evaluate stages in the evolution of our camera engine. Diagrams of typical results are shown in figure 10.

A naive implementation uses just the hard constraints without visibility solving. The results are smooth, but collision through objects and obstructed player views are frequent. A better version uses constraints including visibility computation, but without tolerance settings or predictive measures. We expect that these results are similar to the

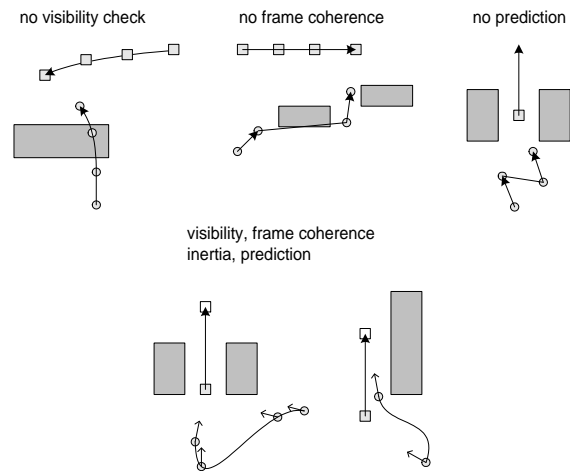


Figure 10: Common artefacts of reactive camera planning, only with the introduction of predictive camera planning can we react to avoid certain situations.

work of Bares and Lester¹, causing "jumping" artefacts and poor frame coherence. Next, we introduce relaxation of constraints. The camera movements are smoother, but still have problems with frame coherence when jumping out of visibility traps. After applying inertia to the camera, so that it prefers to continue along the path it came from, results are less jumpy than before, but we get a ping-pong effect from the change of directions when resolving visibility positions. Finally, estimation routines adjust the camera movement to solved predicted camera states. Now the camera is able to adjust ahead of time to players moving around corners and through objects, and accommodates to situations where the player enters a room, providing a view of the door and its contents before the player is fully inside. In our implementa-

tion, we achieve surprisingly good visual results from state predictions based only on past motion data. In a computer game, the prediction accuracy would depend on what information can be supplied by the action module and simple extrapolation of past data would only be a worst-case scenario.

5.3. Using Level-Of-Detail Geometry

The effects of using coarser geometry for visibility constraints depend on the game scene and interactive freedom of the player. Our helicopter animation that flies through a city allows buildings to be represented as bounding boxes for use on the PVR, creating a substantial frame-rate increase, without detriment to the camera movement. However, those objects which the player may move through (e.g. moving under a table or through a hoop), not only need to share strong similarities in topology, but must also guarantee that the simplified object's volume is a superset of the original object, which prevents the camera from "colliding" with the visualised geometry.

5.4. Fixed Camera Paths

In some cases prior knowledge about a scene is known. There might be a cut-scene in which the player is known to travel along a certain path, or a small enclosed room to which the player is constrained to a limited area. A cut-scene artist can then plan a fixed camera path to express certain visual goals or create dramatic views and effects manually. However, there may be certain aspects of the scene that have been generated at run-time (such as additional characters present) and could not have been accounted for in the game design stage.

In such cases, the camera must adjust for additional possibilities of occlusion, and we may place PVR along the preset camera path or fixed camera setups. For instance, we adjust the camera along the camera path whenever occlusion occurs, or cut to a new pre-setup camera position also defined in the PVR.

6. Concluding Remarks

We have presented what we believe to be the first frame coherent constraint-based camera engine that allows visual goals to be applied on interactive 3d computer games comprising scenes of arbitrary spatial configurations and complexity.

We have made the following observations:

- Visibility satisfaction should be an integral part of any camera system and should be coupled closely to the satisfaction of camera constraints.
- The only way to avoid backing out of a dead-end is to look ahead in time before you go in. Therefore, developing algorithms that estimate the future state of an environment

are necessary for high quality control of the camera. The more effective these predictive algorithms, the more subtle the camera motions will be.

- Using a global best-fitness camera state is not necessarily the best solution for presenting visual goals. Choosing a partially satisfied camera state that lies closer to the current state often results in smoother and more subtle animations.

We conclude our results as follows:

- We have been able to show how a camera engine is to be successfully integrated as part of the dynamic game environment. Dynamic templates are created from conditions set by events, from which constraint-based specifications are formed.
- We have introduced an efficient and highly flexible way to compute visibility constraints for an arbitrary number of points.
- Our camera system works for arbitrary dynamic scenes and spatial complexities of environments. The camera needs no specialised collision information – this is handled effectively and automatically by the visibility constraint computed from the scene geometry.
- We are the first to provide a constraint-solver based on existing camera state and motion characteristics. The methods are fast, consistent, and robust, producing intelligent "nearest-best-fit" frame-coherent camera animations in real-time by reacting to future conditions.

For future work, we are now able to go a step higher in the presentation pipeline. We can solve for visual goals knowing that our low-level constraints will produce coherent results, that they stay in a partially satisfied region rather than jump erratically to global best-fit regions. Using an evolutionary design we can start adding and learning new techniques for camera management, introducing more sophisticated navigational goals and predictive techniques. The goal is to lead to an even more enticing atmosphere created by effective camera work.

Acknowledgements

Thanks to C. Bode and J. von Heinze from e|media for many discussions on the topic of the paper. This work was in part financed by a doctor of scholarship from the Province of Saxony-Anhalt, and by a grant from the multimedia company e|media, Magdeburg.

References

1. W. H. Bares and J. C. Lester. Intelligent multi-shot visualization interfaces for dynamic 3D worlds. In M. Maybury, editor, *Proceedings of the 1999 International Conference on Intelligent User Interfaces (IUI-99)*, pages 119–126, N.Y., Jan. 5–8 1999. ACM Press. [2, 4, 8](#)

2. W. H. Bares, S. Thainimit, and S. McDermott. A model for constraint-based camera planning. In *Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20–22, 2000)*, pages 84–91, Menlo Park, 2000. AAAI Press. 4
3. H. Barwood. Cutting to the chase: Cinematic construction for gamers. At *Game Developers Conference*, 2000. 2
4. J. F. Blinn. Jim blinn's corner: Where am I? what am I looking at? *IEEE Computer Graphics and Applications*, 8(4):76–81, July 1988. 3
5. A. Butz. Anymation with CATHI. In *Proceedings of AAAI-97/IAAI-97*, pages 957–962, Menlo Park, July 27–31 1997. AAAI Press. 3
6. D. Christianson, S. Anderson, L. He, D. Salesin, D. Weld, and M. Cohen. Declarative camera control for automatic cinematography. At *Thirteenth National Conference on Artificial Intelligence*, 1996. 2, 3
7. M. F. Cohen and D. P. Greenberg. The hemi-cube: A radiosity for complex environments. At *held in San Francisco, CA; 22–26 July 1985*, July 1985. 2
8. S. Drucker. *Intelligent Camera Control for Graphical Environments*. PhD thesis, MIT Media Lab, 1994. 2
9. S. M. Drucker, T. A. Galyean, and D. Zeltzer. CINEMA: A system for procedural camera movements. *Computer Graphics*, 26(2):67–70, Mar. 1992. 2
10. S. M. Drucker and D. Zeltzer. Intelligent camera control in a virtual environment. In *Proceedings of Graphics Interface '94*, pages 190–199, Banff, Alberta, Canada, May 1994. Canadian Information Processing Society. 2, 3
11. S. M. Drucker and D. Zeltzer. CamDroid: A system for implementing intelligent camera control. In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 139–144. ACM SIGGRAPH, Apr. 1995. ISBN 0-89791-736-7. 2
12. M. Gleicher and A. Witkin. Through-the-lens camera control. In E. E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 331–340, July 1992. 1
13. N. Halper and P. Olivier. CAMPLAN: A Camera Planning Agent. In *Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20–22, 2000)*, pages 92–100, Menlo Park, 2000. AAAI Press. 2, 4
14. A. Hanson and E. Wernert. Constrained 3d avigation with 2d controllers. *IEEE Visualisation*, pages 175–182, 1997. 1
15. L. He, M. F. Cohen, and D. H. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04–09 August 1996. 2
16. F. Jardillier and E. Languénou. Screen-space constraints for camera movements: the virtual cameraman. *Computer Graphics Forum*, 17(3):175–186, 1998. ISSN 1067-7055. 1
17. P. Karp and S. Feiner. Issues in the automated generation of animated presentations. In *Proceedings of Graphics Interface '90*, pages 39–48, May 1990. 3
18. P. Karp and S. Feiner. Automated presentation planning of animation using task decomposition with heuristic reasoning. In *Proceedings of Graphics Interface '93*, pages 118–127, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society. 3
19. M. Kilgard. Creating reflections and shadows using stencil buffers. At *Game Developers Conference*, 1999. 7
20. J. Mackinlay, S. Card, and G. Robertson. Rapid controlled movement through a virtual 3d workspace. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 171–176, July 1990. 1
21. P. Olivier, N. Halper, J. Pickering, and P. Luna. Visual composition as optimisation. In *AISB Symposium on AI and Creativity in Entertainment and Visual Art*, pages 22–30, Edinburgh, 1999. 4
22. C. B. Phillips, N. I. Badler, and J. Granieri. Automatic viewing control for 3D direct manipulation. In M. Levoy and E. E. Catmull, editors, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 71–74, Cambridge, MA, Mar.–Apr. 1992. ACM Press. 1, 2
23. B. Tomlinson, B. Blumberg, and D. Nain. Expressive autonomous cinematography for interactive virtual environments. In C. Sierra, G. Maria, and J. S. Rosenschein, editors, *Proceedings of the 4th International Conference on Autonomous Agents (AGENTS-00)*, pages 317–324, NY, June 3–7 2000. ACM Press. 2, 3
24. C. Ware and D. Fleet. Context sensitive flying interface. *Interactive 3D graphics*, pages 127–130, 1997. 1
25. C. Ware and S. Osborn. Exploration and virtual camera control in virtual three-dimensional environments. *Interactive 3D graphics*, pages 175–184, 1990. 1
26. H. Zhang. Forward shadow mapping. In G. Drettakis and N. Max, editors, *Rendering Techniques '98*, Eurographics, pages 131–138. Springer-Verlag Wien New York, 1998. 7

4.2 Paper 2: [AVF04]

C. Andújar, P. Vázquez, and M. Fairén: *Way-finder: Guided tours through complex walk-through models*. Proceedings of the Eurographics 2004 Conference (EG 04) 23, 3 (2004), 499-508.

Way-Finder: guided tours through complex walkthrough models

C. Andújar, P. Vázquez, and M. Fairén

Dept. LSI, Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

The exploration of complex walkthrough models is often a difficult task due to the presence of densely occluded regions which pose a serious challenge to online navigation. In this paper we address the problem of algorithmic generation of exploration paths for complex walkthrough models. We present a characterization of suitable properties for camera paths and we discuss an efficient algorithm for computing them with little or no user intervention. Our approach is based on identifying the free-space structure of the scene (represented by a cell and portal graph) and an entropy-based measure of the relevance of a view-point. This metric is key for deciding which cells have to be visited and for computing critical way-points inside each cell. Several results on different model categories are presented and discussed.

1. Introduction

Advances in modeling and acquisition technologies allow the creation of very complex walkthrough models including large ships, industrial plants and architectural models representing large buildings or even whole cities.

These often densely-occluded models present a number of problems related to wayfinding. On one hand, some interesting objects might be visible only from inside a particular bounded region and therefore they might be difficult to reach. On the other hand, walls and other occluding parts keep the user from gathering enough reference points to figure out his location during interactive navigation. This problem becomes more apparent in indoor scenes which often include closed, self-similar regions such as corridors. Finally, architectural and furniture elements can become barriers in collision-free navigation systems. For instance, smooth navigation through turning staircases or narrow passages might require advanced navigation skills.

As a consequence of the above problems, the user may wander aimlessly when attempting to find a certain place in the model, or may fail in finding again places recently visited. Sometimes the user is also unable to explore the whole model or misses relevant places.

One obvious solution to these problems is to provide the user with different navigation aids such as maps showing a

sketch of the scene along with the current camera position. This solution alleviates the problem of disorientation, but still the user can miss important parts. Moreover, automatically generating illustrative maps is not an easy task. Other useful navigation aids such as somehow marking already-visited places are not enough for guaranteeing a profitable exploration.

Another solution is to explore the model following a pre-computed path or a selection of precomputed viewpoints. This path can be provided by the model creator or by an experienced user who already knows the interesting regions of the scene, but this is not always feasible. Moreover, this can also become a disadvantage if we cannot express properly which are the regions that are important for us to visit. In any case, this solution also requires a noticeable user effort during the path definition.

In this paper we present an algorithm for the automatic construction of exploration paths. Given an arbitrary geometric model and a starting position, the algorithm computes a collision-free path represented by a sequence of nodes, each node having a viewpoint, a camera target and a time stamp. The algorithm proceeds through three main steps. First, a cell-and-portal detection method identifies the overall structure of the scene; second, a measurement algorithm

is used to determine which cells are worth visiting, and finally, a path is built which traverses all the relevant cells.

The rest of the paper is organized as follows. Section 2 reviews previous work on automatic path generation and cell-and-portal detection. Section 3 presents a characterization of the properties that we consider suitable for a camera path. Section 4 gives an overview of our approach. Section 5 presents our algorithm for the automatic generation of cells and portals and Sections 6 and 7 explain how the more relevant cells are determined and how the exploration path is built respectively. We present some results in Section 8 and conclude our work pointing some lines of future work in Section 9.

2. Previous Work

Motion planning has been extensively studied in robotics, computational geometry and related areas for a long time. However, it is still considered to be a difficult problem to solve in its most basic form, e.g., to generate a collision-free path for a movable object among static obstacles. As stated by Canny [1], the best known complete algorithm for computing a collision-free path has complexity exponential in the number of degrees of freedom of the robot or the moving object. Good surveys can be found in [2] and [3].

Some approaches for motion planning present algorithms formulated in the configuration space of a robot. The configuration space (also known as *C-space*) is the set of all possible configurations of a mobile object. Ito presents two approaches, the first one [4] computes a decomposition of the *C-space* and searches the graph connecting collision-free areas of the decomposition for a correct path. The second one [5] divides the search algorithm in two levels: a global search and a local search.

Other sorts of algorithms are based on randomized motion planning. Li *et al.* [6, 7] take input from the user and predict the location where the avatar should move to. However, this approach has only been used for navigation in simple environments due to its high running time. Salomon *et al.* [8] present an interactive navigation system that uses path planning. The path is precomputed using a randomized motion planning with a reachability-based analysis. It computes a collision-free path at runtime between two specified locations. However, their system still needs more than one hour to compute a roadmap for relatively simple models (ten thousand polygons) and sometimes the results are unnatural paths. Kallmann *et al.* [9] present a new method that use motion planning algorithms to control human-like characters manipulating objects which allow up to 22 degrees of freedom.

In our approach, the configuration space depends on the spatial structure of the scene and we want to explore it by means of cells and portals, so the graph we need is completely different, we need a *cell-and-portal* graph.

A cell-and-portal graph (CPG) is a structure that encodes the visibility of the scene, where nodes are cells, usually rooms, and edges are portals which represent the openings (doors or windows) that connect the cells. The construction of a CPG is commonly done by hand, so it is a very time consuming task as the models become more and more large and complex. The automatic generation of portals and cells is therefore a very important issue. There are few papers that refer to the automatic determination of portal-and-cell graphs, and most of them work under important restrictions. Teller and Séquin [10] have developed a visibility preprocessing suitable for axis-aligned architectural models. Hong *et al.* [11] take advantage of the tubular nature of the colon to automatically build a cell graph by using a simple subdivision method based on the center-line (or skeleton) of the colon. To determine the center-line, they use the distance field from the colonic surface. Haumont *et al.* [12] present a method that adapts the 3D watershed transform, computed on a distance-to-geometry sampled field. However, their method only works on cells free of objects, and therefore these have to be removed previously by hand.

3. Camera path characterization

Given a geometric model, there is an infinite number of paths exploring it. In order to compute paths algorithmically we have to identify which are the properties that distinguish a suitable path from non-useful ones. The following list presents the main properties users might expect from a camera path.

- *Collision-free*

Ideally, a camera path should be free from collisions with scene objects. However, this is not always feasible since the input scene might contain interesting parts bounded by closed surfaces which will be impossible to reach using this criterion strictly. Therefore we require our paths to not cross any wall unless it is the only way to enter a cell bounded by a closed surface.

- *Relevant*

A good path must show the user the most *relevant* parts of the model while skipping non relevant or repetitive parts. Relevance is a subjective quality that depends on user interests, but requiring the user to identify and mark relevant objects would compromise the scalability of our approach. As a consequence, a metric for estimating relevance is required. One contribution of this paper is the use of entropy-based measurements for quantifying the relevance of a given viewpoint.

- *Non-redundant*

Ideally, a camera following the path should visit each place only once. Again, this is often not possible e.g. traversing the same corridor many times can be the only way to visit all relevant rooms. We therefore require our algorithm to avoid already visited places whenever possible.

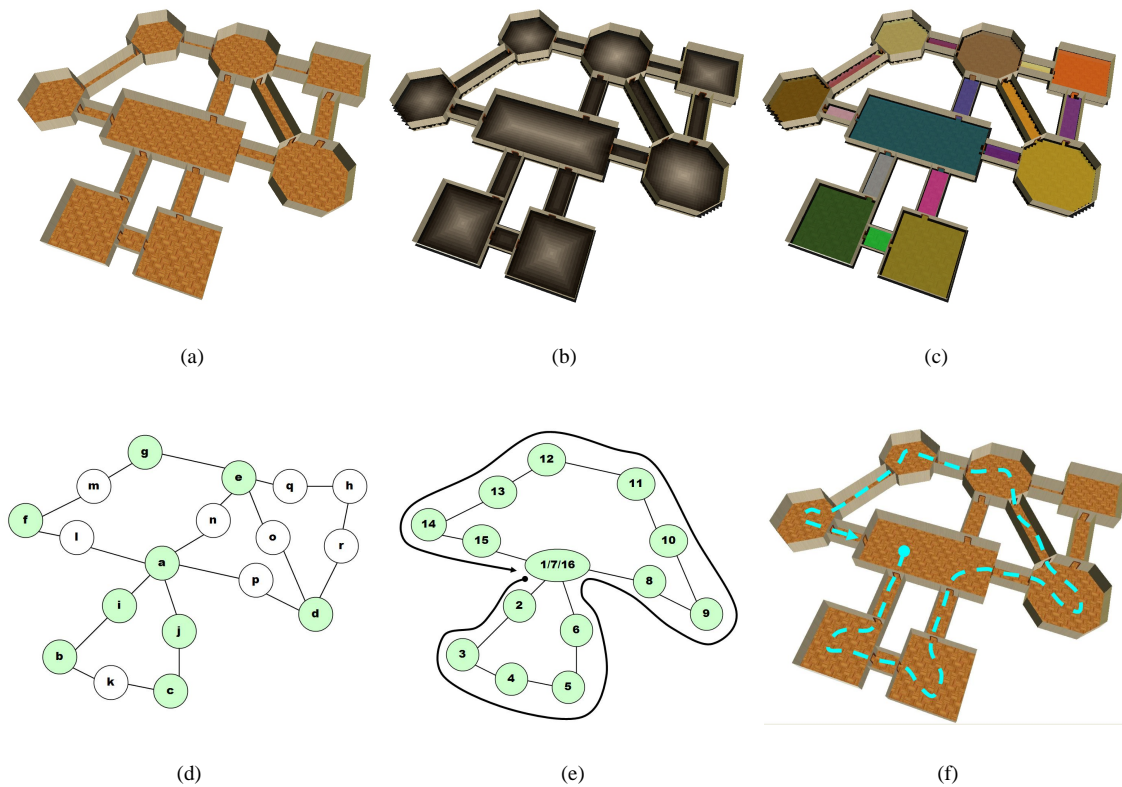


Figure 1: Overview of our approach. (a) Input scene (furniture and ceiling not shown); (b) Distance-to-geometry field computed over the 3D grid (only one slice is shown); (c) Cells detected with random color (note that corridors are also identified as cells); (d) Cell-and-portal graph embedded in the model space; cells are labeled according to relevance measure; (e) High-level path computed as a sequence of cells; visited cells is a superset of relevant ones; (f) Final path after smoothing (camera target not shown).

- *Uncoupled target*

In most online navigation systems, the camera target is defined in accordance with the forward direction of the viewpoint as this facilitates the camera control. However, precomputed paths do not benefit from this limitation. Uncoupling the camera target from the advance direction is often desirable because it allows the user e.g. to watch the paintings on the ceiling of a room while crossing it.

- *Ordered*

This property is closely related to the non-redundancy criterion. The path should not leave a room unless all the relevant details it contains have been visited.

- *Self-adjusting speed*

In addition to let the user modify the camera speed during the reproduction of the path, it is also convenient to define the path so that the speed is defined in accordance with the relevance of the part of scene being seen. This implies that the speed increases while traversing open spaces with distant details or when walking through already visited

places. Similarly, the speed decreases while approaching relevant objects.

- *Smooth*

The path creator should try to avoid abrupt changes in speed, camera position and camera target.

4. Algorithm overview

Our algorithm receives as input an arbitrary walkthrough model and a starting camera position and computes a collision-free path represented by a sequence of nodes, each node having a viewpoint, a camera target and a time stamp.

The algorithm proceeds through three main steps (see Figure 1).

First, we identify the free-space structure of the scene by computing a cell and portal graph $G = (V, E)$ over a grid decomposition (Section 5). Our cell and portal graph differs from the ones used for visibility computation in that we do not need to classify the scene geometry against the cells nor

do we need to compute the exact shape of the portals. In fact, our cells are simply represented by a collection of voxels and for each portal we just need a single way-point. This cell decomposition allows the algorithm to produce paths with minimum redundancy where cells are visited in a natural way, the portals being suitable waypoints.

In a second step (Section 6) we use an entropy-based measurement algorithm to identify the cells in V that are worth visiting (*relevant* cells). This step filters out non-interesting cells and also ensures the robustness of the algorithm against an over-decomposition of the scene into cells due to geometric noise.

The last step builds a camera path which traverses all the relevant cells and visits the more interesting places inside each cell (Section 7). This task is accomplished at two levels. We first decide *in which order the relevant cells should be visited* by computing a path H over the cell-and-portal graph. We call H the *high-level* path, which is just an ordered sequence of cell identifiers and portals connecting adjacent cells. For this task the algorithm must find the shortest path traversing all the relevant cells while minimizing the traversal of non relevant cells and repeating cells. At this point our path contains only a few waypoints which correspond to the portals connecting adjacent cells on the high-level path. The next task is to decide how to refine the path *inside each cell*. This is accomplished by computing a sequence of waypoints for visiting each cell from an entry-point to an exit-point. Again the entropy-based measure is used for deciding both the waypoints and the best camera target at each viewpoint. Note that both entry and exit points are just the center of the portals connecting the current cell with the previous cell and the next cell respectively. Finally, a simple postprocess smoothes the path and adjusts the speed in accordance to the precomputed relevance of the viewpoints.

5. Automatic portal and cell detection

The creation of the cell-and-portal graph pursues two aims. On the one hand, the cell decomposition provides a high-level unit for evaluating the relevance of a region and for deciding whether this region should be visited or not. Moreover, this decomposition allows for solving the problem of finding collision-free paths considering only one cell at a time. On the other hand, the portal detection provides a first insight into the final path because portals are natural waypoints.

Our approach for computing the cell-and-portal graph is based on conquering quasi-monotonically decreasing regions on a distance field computed on a grid. The cell detection is organized in successive stages explained in detail below. First, we build a binary grid separating empty voxels from non-empty ones. Next we approximate the distance field using a matrix-based distance transform. Then we start an iterative conquering process starting from the voxel

having the maximum distance among the remaining voxels. During this process, all conquered voxels are assigned the same cell ID. A final merge step eliminates small cells produced by geometric noise. Finally, faces shared by voxels with different cell ID's are detected and portals are created at their centers.

5.1. Distance field computation

The first step converts the input model into a voxel representation encoded as a 3D array of real values. Voxels traversed by the boundary of the scene objects are assigned a zero value whereas empty voxels are assigned a $+\infty$ value. This conversion can be achieved either by a 3D rasterization of the input model or by a simultaneous space subdivision and clipping process supported by an intermediate octree [13].

The next step involves the computation of a distance field (Figure 1-b). The distance field of an object is a 3D array of values, each value being the minimum distance to the encoded object [14]. Distance fields have been used successfully in generating cell-and-portal graph decompositions [12]. The distance field we consider here is unsigned. Distance fields can be computed in a variety of ways (for a survey see [14]). We approximate the distance field using a *distance transform*. Distance transforms can be implemented through successive dilations of the non-empty voxels and more efficiently by a two-pass process. The Chamfer distance transform [14] performs two passes through each voxel in a certain order and direction according to a distance matrix. The local distance is propagated by the addition of known neighborhood values provided by the distance matrix. In our implementation we use the 5x5x5 quasi-euclidean chamfer distance matrix discussed in [14]. Indeed, our experiments show that computing the distance field on a horizontal slice of the voxelization (using the central 5x5 submatrix) leads to better cell decompositions as it limits the influence of the floor and ceiling and it is less sensitive to geometric noise caused e.g. by furniture. Note that the maxima of the distance transform (white voxels in Figure 1-b) can be seen as an approximation of the Medial-Axis Transform.

5.2. Cell generation

The cell decomposition algorithm visits each voxel of the distance field and replaces its unsigned distance value by a cell ID. We use negative values for cell ID's to distinguish visited voxels from unvisited ones. The order in which voxels are visited is key as it completely determines the shape and location of the resulting cells.

The order we propose for labeling cells relies on a conquering process starting from the voxel having the maximum distance among the remaining unvisited voxels. This local maximum initiates a new cell whose ID is propagated using a breadth-first traversal according to the following propagation rule. Let v be the voxel being visited, and let D_v be the

```

procedure cell_decomposition
  cellID = -1
  S = sort_voxels()
  while not_empty(S) do
    (i,j,k) = pop_maximum(S)
    if grid[i,j,k] > 0 then
      expand_voxel(i,j,k,cellID)
    end
    cellID = cellID - 1
  end
end

```

Figure 2: Cell decomposition algorithm

distance value at voxel v . The current cell ID is propagated from v to a face-connected neighbor v' if $0 < D_{v'} \leq D_v$, i.e. the distance value at v' is positive but less or equal than the distance at v . The propagation of the cell ID continues until the whole cell is bounded by voxels having either negative distance (meaning already visited voxels), zero distance (non-empty voxels) or positive distance greater than the voxels at the cell boundary. Then, a new unvisited maximum is computed and the previous steps are repeated until all non-zero voxels have been assigned to some cell (Figure 2).

Furniture and other scene objects might exert a strong influence on the distance field, causing many local maxima to appear and therefore producing an over-segmentation of the cell decomposition. A straightforward solution could be to remove by hand all furniture elements before the model is converted into a voxelization, which is the solution adopted in [12]. The solution we propose is to relax the propagation process by including a decreasing tolerance value in the propagation rule: the ID is propagated from v to v' if $0 < D_{v'} \leq D_v + \epsilon$, where ϵ vanishes to zero as the cell grows. The consequence of this aging tolerance is that small variations of the distance field near the cell origin do not impact their propagation. This variation is less sensitive to noise than a watershed transform considering simultaneously all local maxima [12]. The connectivity used during the propagation process is 4-connectivity in 2D and 6-connectivity in 3D. A two-dimensional propagation suffices e.g. when the camera height (with respect to the floor) remains constant during the path.

5.3. Cell merging and portal detection

A cell merging process further improves the cell decomposition by merging uninteresting cells. Let $|A|$ be the size of cell A , measured as the number of voxels, and let $Portal(A, B)$ be the number of voxels shared by cells A and B . We use the following merging rules: (a) if $|A|$ is smaller than a given minimum size then the cell is merged with the cell sharing the large number of boundary faces with A ; if no such a cell exist (i.e. A is bounded by 0-distance voxels) then the cell A is removed; (b) if $Portal(A, B)$ is greater than a maximum

portal size, then cells A and B are merged into a single cell. The results shown in Section 8 have been computed using only the first rule.

The graph nodes in V correspond to the identified cells and the graph edges in E correspond to links between adjacent cells. Besides the graph connectivity, each cell is represented by a collection of face-connected empty voxels and a graph edge connecting two cells is represented by the collection of portals shared by the two cells. Portal detection is straightforward and requires a single traversal of the voxels identifying faces shared by voxels with different IDs. Portals correspond to connected components of shared voxels. Each portal is assigned a single point that can be computed as the portal center. An alternative which works better for non-planar portals consists in keeping the distance field values during the cell generation process (instead of re-using these values for storing the cell IDs) and compute the portal representant as the voxel with the highest value on the distance field (i.e. the point on the portal farthest from the nearby geometry). These points are candidates for waypoints in case the path has to cross the portal for going from one cell to another.

6. Identifying relevant cells

Once we have determined the graph of portals and cells, the following step is to determine which are the cells that are worth visiting in the model.

The data structure built to determine the cell-and-portal graph is also useful to give a set of points inside each cell. Given this set of points, we can determine if the cell is relevant by measuring the amount of information that can be seen from each point of the set. In order to compute the amount of information we use an entropy-based measure, dubbed viewpoint entropy, which has been successfully applied to determine the best view of objects and scenes [15]. We measure and store the point of maximum entropy for each cell and then choose those cells that have a higher relevance. These selected cells will be the *relevant cells* to be visited.

6.1. Viewpoint Entropy

Viewpoint entropy is a measure based on Shannon entropy [16]. It uses the projected areas of the visible faces on a bounding sphere of the viewpoint to evaluate how much of the visible information can be seen from the point. For a single view, we only need to render the scene into an item buffer from the viewpoint. Then, the buffer is read back and we sum the area of each visible polygon (actually as we should render to a sphere, we weigh each pixel by its subtended solid angle, to calculate entropy properly). Then, the relevance is computed using the following formula:

$$H_p(X) = - \sum_{i=0}^{N_f} \frac{A_i}{A_t} \log \frac{A_i}{A_t},$$

where N_f is the number of faces of the scene, A_i is the projected area of face i , A_t is the total area covered over the sphere, and A_0 represents the projected area of background in open scenes. In a closed scene, or if the point does not *see* the background, the whole sphere is covered by the projected areas and consequently $A_0 = 0$. The maximum entropy is obtained when a certain point can *see* all the visible faces with the same relative projected area A_i/A_t . To cover all the surrounding space we need six projections (similarly to the cube map construction process).

6.2. Relevance cell determination

To detect the important cells, we select a set of viewpoints inside each cell. The candidate points are given by the cell detection algorithm (for example one per voxel if voxel size is small enough). The viewpoint entropy of each candidate point is evaluated and stored in order to select the best one, whose entropy will indicate the relevance of the cell. Usually, the cells detected in the first step will be relatively free of objects, and large occluders will naturally determine new portals and cells. Throughout the process of determination of the relevance of a cell, we can store the visible projected areas of each face for each evaluated viewpoint. Then, we can determine the best set of views by iteratively selecting the best one, marking the already visited faces, and recomputing the entropy values for the rest of the views only taking into account the not yet visited faces [15]. If almost all the visible faces were visible from the best view, this means that there are no large occluders in our cell. Otherwise we select more than one important point in the cell for a future visit. Note that the example in Section 8 only yields one viewpoint per cell, as the selected points are placed relatively close to the center and therefore they capture much information. Notice that if the discretization is roughly the same, the camera will be *attracted* by regions of high number of polygons. Otherwise, we can set an importance value to the polygons we consider interesting (such as the ones belonging to statues). In the examples presented here we have not considered texturing, but this can be addressed using a region growing segmentation and posterior color coding of the regions to include the resulting texture in our measure as different polygons, as detailed in Vázquez *et al.* [15]. Viewpoint entropy has also been used for automatic interactive navigation in indoor scenes [17]. Unfortunately, the lacking of knowledge of the general structure of the model makes it difficult to ensure that the camera will pass through all the relevant cells. An entropy-based measure has also been presented and used to automatically place light sources [18].

7. Path construction

With the information collected from the previous two steps, we can build a minimal length path through the graph that visits all relevant cells. Our objective is not only to determine the path that covers all interesting cells but to determine at each moment which is the suitable camera position in order to see the highest amount of information of the scene.

7.1. High-level path

The first step on the path construction is to decide in which order the path will visit the relevant cells. We compute a high-level path H over the cell-and-portal graph which is the shortest path traversing all the relevant cells from the initial point given by the user.

Given the set of relevant cells to visit and the initial cell, the problem of finding the shortest path traversing all the relevant cells is similar, but not equal, to the traveling salesman problem (TSP) which is an NP-complete problem [19]. We use a backtracking algorithm optimized by discarding partial solutions when they are longer than an already found solution. When the search is finished, we have a group of solutions that are minimal on its length (number of nodes traversed), and we choose the one with minimal node repetition. The cost of this algorithm is not enlarging the total cost of the approach much since the models usually don't have more than 50 cells. Nevertheless the cell merging process in phase 1 can be adjusted to limit the number of cells.

7.2. Low-level path

The *low-level* path can be computed just after the high-level path generation. Once we know which cells have to be visited and which is the ordering, we get an entrance and an exit point for each cell. This, together with the best viewpoint (or viewpoints) of each cell allows us to build a smooth path. To build this path we perform two steps:

1. Path detection
2. Path approximation

As we do not know in advance which is the geometry of the cell and we only get a set of points inside it, the determination of a free-from-obstacles path must be done accurately. Given the set of points corresponding to voxels inside a cell, we build a graph where the nodes are the points and the edges the connectivity of the points to the neighbors. This can be carried out very fast. Then, we apply an A* algorithm [20] to search the best path from the entrance point to the best point of the cell, and we apply it again to reach the exit from the best point. For complex cells (i.e. the cells that generate more than one best points) this is applied iteratively until reaching the exit. In these cases, the iteration is also applied from the exit point to the entrance point, generating an alternative path which could be shorter than

the previous one. If this *backwards* path is shorter, we will choose it reverting the point order.

The generated path is a polyline that is not necessarily smooth. To avoid sharp moves through the exploration process, we relax the path by using an interpolation based on Hermite curves [21]. We set a control point every two points of the path and build a smooth path that goes from the entrance to the exit. At each control point we use as tangent direction the vector joining the current point with the next path point. Note that it is better not to enforce C^1 continuity on the path at the best view point. The best point is supposed to show a high amount of interesting information, so the walkthrough will stop there and the camera will rotate to allow the user to see all the important information. In Figure 3 we can see an example of a path inside a cell. As the set of camera positions is very dense, we have only drawn one out of five camera positions. The yellow line indicates the orientation of the camera at these positions.



Figure 3: An example path inside a cell. Camera positions are shown in red and the orientations appear as yellow lines.

7.3. Complete Walkthrough

After the construction of the path, we want to determine which are the camera orientations that better show the scene during the navigation.

In a similar process than the one that determines the best point inside each cell, we place a camera at each point of the path and, for each point, we evaluate the amount of information that can be seen at different orientations and choose the orientations that will yield better results. This is computed almost interactively. We set some reasonable constraints to camera moves in order to build a smooth path.

- **Limited rotation:** The camera must be oriented forward, we do not allow rotations of more than 30-40 degrees from the walking direction in order to maintain a normal

movement sensation during the walkthrough. If the camera were allowed to point backwards, the user could feel uncomfortable.

- **Correct orientation at endpoint.** People usually look forward when traversing a portal. We simulate this by limiting the rotation of the camera when it is reaching the exit point. When the camera is close to the way out, it smoothly starts turning back to the walking direction, and we ensure that it is at the correct orientation before crossing the portal.

For each cell the path is built in the following way. We place the camera at the entrance point of the cell and pointing towards inside the cell. Then, the best new camera orientation is computed by evaluating the possible new orientations (these measures are calculated on the back buffer), we allow only small rotations in order to make the movements smooth. For a given point and camera orientation, five different views are inspected, as depicted in Figure 4.

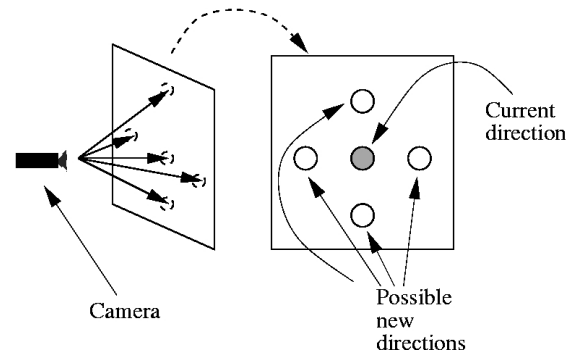


Figure 4: The possible changes of orientation of the camera at each step

To decide which of the orientations is the best, we take into account the amount of information visible from each camera configuration, as well as the history of the visited regions. This can lead to a problem when there is a very complex region in a cell, because the camera would be always pointing there. In order to avoid this, we keep track of the visited faces. For each view, when we analyze the amount of visible information we only take into account the faces that have not been visited yet. However, as the path will contain one or two hundred different positions at each cell, this could be a problem if we considered a polygon as visited if it had appeared in a single view, because it could cause the camera to change the orientation continuously. As we want the user to be able to see the environment properly, we have implemented a pseudo aging policy. We only consider a face visited when it has been seen at least in 20 different views, then it is marked and it will not be considered again. This strategy ensures that the regions of higher information will be visited and that the user will be able to stare at them calmly.

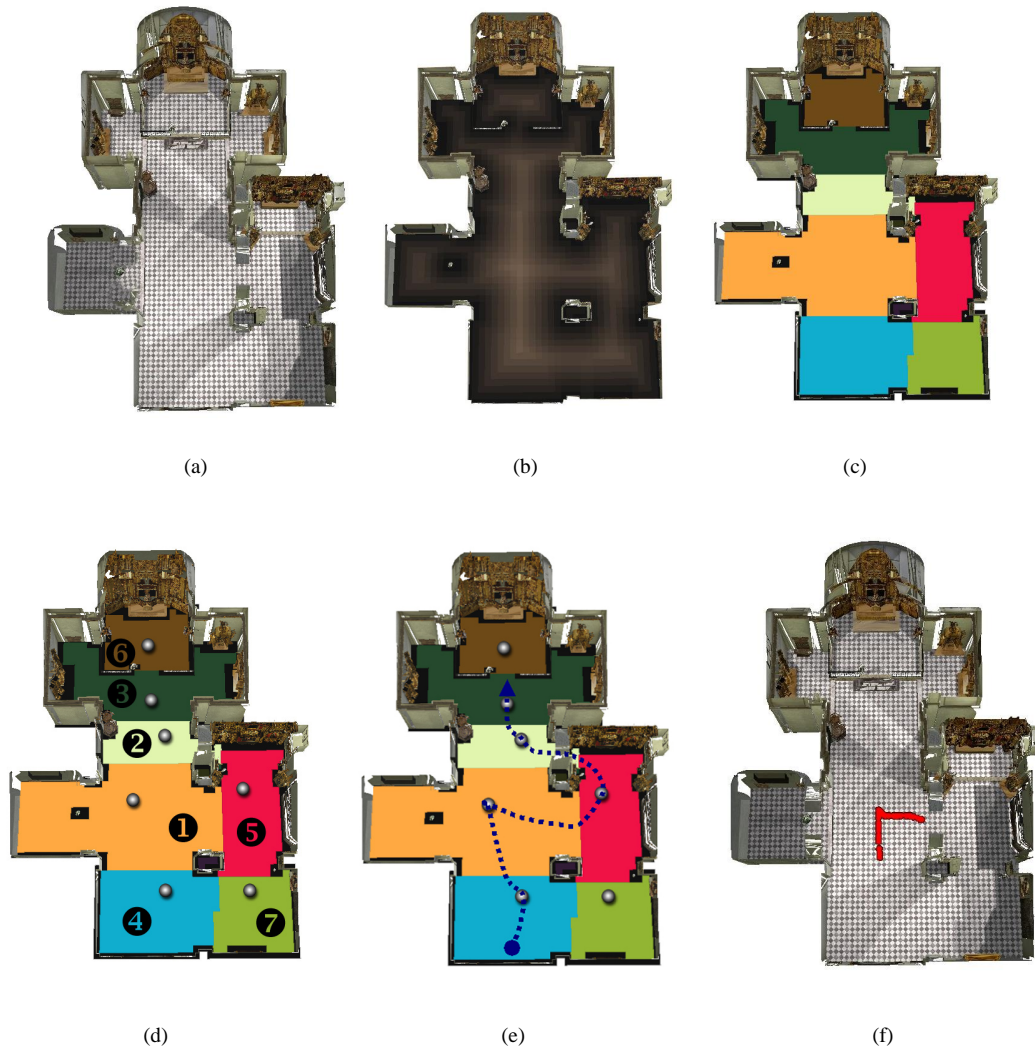


Figure 5: Results in the church model. (a) Top view of the original model. (b) Computed distance field on a $128 \times 128 \times 128$ grid. (c) Final cells detected after the merging process. (d) Ordered cells based on their entropy. (e) High level path through the 5 most interesting cells. (f) Computed low-level path for the most interesting cell.

8. Results

We have presented a complete approach for automatically generating guided tours through complex walkthrough models. In contrast to other approaches, our method is completely automatic, the only input really required is an initial point.

Images of the whole process appear in Figure 5. In Figure 5-a we show the plan of the original model. The computation of the distance field map appears in Figure 5-b. After the distance field computation, the cell and portal generation detects a set of cells that are then refined through the

merge process. The result of the merge for this example is shown in Figure 5-c. With this information we can proceed to compute the relevance of each cell. This generates an ordering between the cells that is depicted in Figure 5-d. After the cell evaluation, we choose a subset of cells with high entropy. In this case the threshold chosen selects cells 1 to 5, and the starting point of the path is at cell 4, the entrance of the church. Then, a high level path is calculated using the backtracking method explained in Section 7.1. The computed high level path is shown in Figure 5-e. For each cell, a low level path is computed from the entrance point to the best view of the model. As an example, we show the path

corresponding to the most important cell in Figure 5-f. Note that the generated path has almost 200 camera positions, so we have only shown one out of 5 camera positions (with their corresponding orientations) for the sake of clarity. As we have commented, we do not force continuity on the best viewpoint, as at this point, the camera rotates to show the information all around that was not already seen during the previous walkthrough positions.

The total computation time was 10 minutes and 40 seconds on a 2GHz PIV with a GeForce Ti graphics card and 512Mb of memory with a model (the church model) of 63312 polygons. The bottleneck is the cell relevance evaluation process and the low-level path calculation because both require rendering the scene multiple times, which could benefit from the portal-and-cell graph if we used this for portal culling. More results can be found in <http://www.lsi.upc.es/~virtual/EG2004.html>

In our current implementation the output of our algorithm can be used in several ways. The *full-auto* mode consists in the reproduction of the path by letting the camera follow the precomputed viewpoints and targets. The *guided-tour* variation would let the user look around during the navigation by allowing him or her to control the target but not the viewpoint. Finally, in the *free* mode the path nodes are rendered as arrows oriented along the direction of the next waypoint.

9. Conclusions and Future Work

We have presented a fully automatic system for the generation of walkthroughs inside closed environments that can be segmented using a cell-and-portal approach. The method can be useful as a way to automatically create visits of monuments or presentations of buildings, and can also be a good tool in the context of interactive systems as a first constrained path to help the interactive user navigate an environment.

As future work we want to introduce a hierarchical structure, concretely an octree representation to optimize the cell detection process. Moreover, we would like to further limit the effects of the furniture in the cell detection algorithm. An interesting issue would be to be able to compute cells in outdoor sparse scenes.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Technology under grants TIC2001-2226 and TIC2001-2416.

References

- [1] John F. Canny. *The complexity of robot motion planning*. MIT Press, 1988. 2
- [2] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. 2
- [3] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, September 1992. 2
- [4] P. Isto. Path planning by multiheuristic search via sub-goals. In *Proceedings of the 27th International Symposium on Industrial Robots*, pages 721–726, 1996. 2
- [5] P. Isto. A two-level search algorithm for motion planning. In *Proc. IEEE International Conference on Robotics*, pages 2025–2031, Aut 1997. 2
- [6] T. Y. Li, J. M. Lien, S. Y. Chiu, and T. H. Yu. Automatically generating virtual guided tours. In *Proc. of Computer Animation*, pages 99–106, 1999. 2
- [7] T. Y. Li and H. K. Ting. An intelligent user interface with motion planning with 3d navigation. In *Proc. IEEE VR*, 2000. 2
- [8] Brian Salomon, Maxim Garber, Ming C. Lin, and Dinesh Manocha. Interactive navigation in complex environments using path planning. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 41–50. ACM Press, 2003. 2
- [9] Marcelo Kallmann, Amaury Aubel, Tolga Abaci, and Daniel Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. *Computer Graphics Forum*, 22(3), 2003. 2
- [10] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics*, 25(4):61–68, 1991. 2
- [11] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He. Virtual voyage: Interactive navigation in the human colon. *Computer Graphics*, 31(Annual Conference Series):27–34, 1997. 2
- [12] Denis Haumont, Olivier Debeir, and François Sil-lion. Volumetric cell-and-portal generation. *Computer Graphics Forum*, 22(3):303–312, September 2003. 2, 4, 5
- [13] Carlos Andujar, Pere Brunet, and Dolors Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics*, 21(2):88–105, 2002. 4
- [14] M. Jones and R. Satherley. Using distance fields for object representation and rendering. In *Proc. of the 19th annual Conference of Eurographics (UK chapter)*, London, pages 37–44, 2001. 4
- [15] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Automatic view selection using viewpoint entropy and its application to image-based modeling. *Computer Graphics Forum*, 22(4):689–700, Dec 2003. 5, 6
- [16] E.C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July–October 1948. 5

- [17] P.-P. Vázquez and M. Sbert. Automatic indoor scene exploration. In *International Conference on Artificial Intelligence and Computer Graphics, 3IA*, Limoges, May 2003. [6](#)
- [18] S. Gumhold. Maximum entropy light source placement. In *Proceedings of the Visualization 2002 Conference*, pages 275–282. IEEE Computer Society Press, October 2002. [6](#)
- [19] K. Thulasiraman and M. N. S. Swamy. *Graphs: theory and algorithms*. John Wiley & Sons, Inc., 1992. [6](#)
- [20] Steve Rabin. *"AI Game Programming Wisdom"*. Charles River Media, March 2002. [6](#)
- [21] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics. Principles and Practice. Second Edition*. Addison-Wesley, 1990. [7](#)

4.3 Paper 3: [KKS 05]

Azam Khan, Ben Komalo, Jos Stam, George Fitzmaurice and Gordon Kurtenbach *Hovercam: interactive 3d navigation for proximal object inspection*. In Proceedings of the 2005 symposium on Interactive 3D graphics and games (SI3D 05) (New York, NY, USA, 2005), ACM Press, pp. 73-80.

HoverCam: Interactive 3D Navigation for Proximal Object Inspection

Azam Khan, Ben Komalo, Jos Stam, George Fitzmaurice, Gordon Kurtenbach

Alias

210 King Street East, Toronto, Ontario, Canada
{ akhan | bkomalo | jstam | gf | gkurtenbach }@alias.com
www.alias.com/research

Abstract

We describe a new interaction technique, called *HoverCam*, for navigating around 3D objects at close proximity. When a user is closely inspecting an object, the camera motions needed to move across its surface can become complex. For tasks such as 3D painting or modeling small detail features, users will often try to keep the camera a small distance above the surface. To achieve this automatically, HoverCam intelligently integrates tumbling, panning, and zooming camera controls into a single operation. This allows the user to focus on the task at hand instead of continuously managing the camera position and orientation. In this paper we show unique affordances of the technique and define the behavior and implementation of HoverCam. We also show how the technique can be used for navigating about data sets without well-defined surfaces such as point clouds and curves in space.

Categories and Subject Descriptors: I.3.6 [Computer Graphics]: Methodology and Techniques – Interaction Techniques; H.5.2 [Information Interfaces And Presentation (HCI)]: User Interfaces – Interaction styles, Input devices and strategies.

Additional Keywords and Phrases: interaction techniques, camera controls, 3D navigation, 3D viewers, 3D visualization.

1 Introduction

The most commonly used operation in 3D computer graphics and animation software is camera movement. Users often move the camera to help them sense the 3D properties of a model or animation or while performing modifications. When working at close proximity to an object like, for example, when painting details on an object, the camera must often be moved to see neighboring surfaces. However, despite the heavy usage of camera tools in 3D content creation software, the industry standard zoom, pan, and tumble tools have been the primary camera controls offered to users for over a decade.

For keyboard and mouse based user interfaces, interactive 3D camera control is fundamentally difficult because the task involves controlling the six distinct degrees of freedom (DOF) (translation x,y,z and rotation α,β,χ) of the virtual camera with just two DOF of mouse input. The simplest approach is to assign the two DOFs of the mouse to two different DOFs of the camera, at different times. However, more sophisticated approaches that emulate real world behaviors, or better match the task at hand and/or the skills of the user, have largely replaced the simple approach. For example, the industry standard zoom, pan, and tumble tools reflect typical methods of controlling physical camera from the film production industry. Even more sophisticated camera control techniques take into account information about the scene. For example, 3D video games often have a *walking* metaphor of camera motion. This metaphor suggests many things: there is a ground plane, the viewpoint is somewhat above the ground, camera rotation is egocentric, there is notion of which way is “up”, etc. These constraints simplify camera motion from a general 6 DOF problem to almost a 2 DOF problem. Further constraints, such as collision detection, prevent the camera from passing through walls, characters, and objects in the scene. This entire set of constraints is needed to convey the *walking* camera metaphor.

The camera metaphor we explore in this paper is navigation around 3D objects at close proximity. For this task, we would like to move around the object while maintaining a fixed distance from the surface and while keeping the object roughly centered in the field of view. We call this metaphor *object inspection*. As with the *walking* metaphor described above, we attempt to use as many constraints as possible, given by the context of our metaphor, to simplify the number of controls needed to navigate through space during typical object inspection.

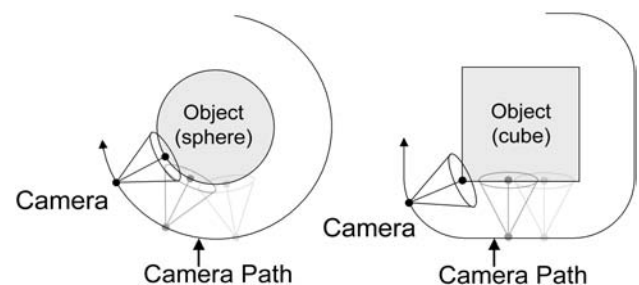


Figure 1. Desired HoverCam motion over a sphere and a cube (shown in profile).

2 HoverCam

Figure 1 shows the type of camera motion behavior we would like for a simple sphere and cube. Note that given the constraints of (1) keeping the camera a fixed distance from the surface and (2) relatively normal to the surface effectively creates a shell around the object being observed, on which the camera can be positioned. Around curved surfaces, the camera follows the surface but at a fixed distance. Around corners, the camera turns to smoothly move toward a neighboring surface. Finally, around flat surfaces, like the side of the cube, the camera pans as expected to keep the underlying surface facing the viewpoint.

Camera paths like these can be achieved using the traditional separate pan, zoom, and tumble tools but at the cost of constantly switching between the three tools. Also, as these tools do not typically perform any collision detection, users may end up in awkward locations inside the object, looking away from the object, or at great distances for the object. For example, in Alias' Maya software, the system is placed in camera mode by holding down the alt-key. Dragging with the left mouse button then tumbles the camera (rotates about the current look-at point). Dragging the middle mouse button pans the camera (translates the eye and the look-at point) and dragging with both left and middle mouse buttons performs zooming (moving the eye toward or away from the look-at point). Releasing the alt-key stops the camera tool and reselects the user's previous tool.

A smooth camera path around the outside of an object is simply not achievable with these separate tools. To keep a point of interest on the surface of the object near the center of the view, the user must always overshoot, switch tools, correct the view with another tool, overshoot again, and so on.

2.1 Basic HoverCam Algorithm

A smooth camera path can be achieved with a trajectory algorithm loosely based on the model of a satellite orbiting an object with a gravity field (see Figure 2). The steps performed are:

- (a) apply user input to the *eye* point E_0 (current camera position) and *look-at* point L_0 , to create E_1 and L_1 ,
- (b) search for the *closest* point C on the object from the new eye position E_1 ,
- (c) turn the camera to look at C , and,
- (d) correct the distance δ_1 to the object to match the original distance to the object δ to generate the final eye position E_2 .
- (e) clip the distance traveled (discussed in Section 2.5).

This algorithm, in effect, selectively combines the operations for zooming, panning, and tumbling during a single mouse drag. This has the advantage that HoverCam only requires a single button mouse, pen-press, or a single finger press on a touch screen to apply camera motion. In contrast, as mentioned earlier, standard zoom, pan, and tumble tools typically require multiple buttons to switch between the operations for zooming, panning and tumbling. Furthermore, to achieve HoverCam motion with the traditional separate tools would require ongoing switching of the tools to continually correct the camera motion to follow the surface.

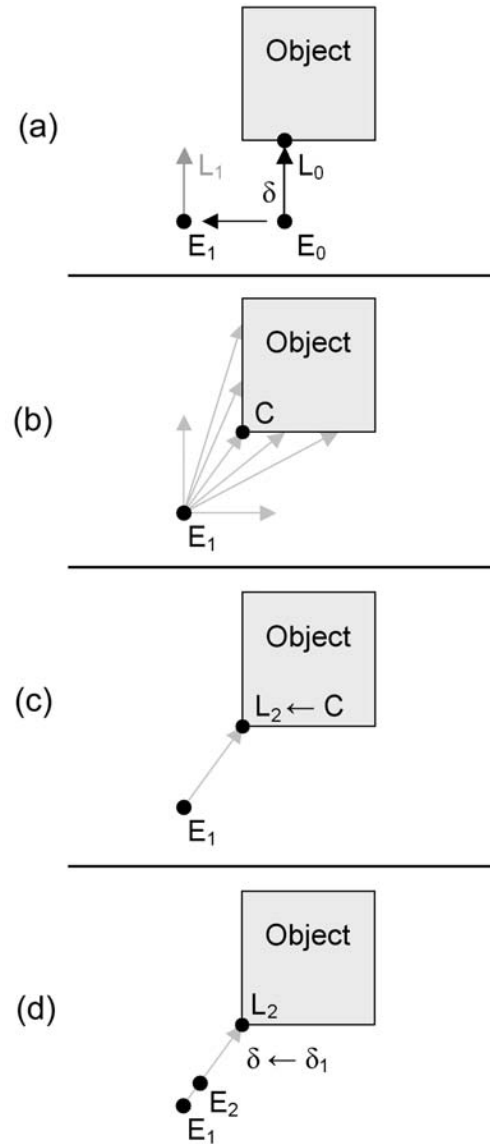


Figure 2. Basic Camera Update Rules. (a) move eye based on user input, (b) look for C , closest point on object, (c) turn camera to C , and (d) correct distance.

Using HoverCam has the feeling of hovering above the object. Figure 3 shows two sets of screen images showing the user's perspective as HoverCam is being used to inspect a cube and a cylinder, maintaining a consistent scale and distance from the object. Note how HoverCam pans on the side of the cube, and turns about the corner of the cube. Also note that on the cylinder, the camera pans along the shaft, turns smoothly to the end disc, and pans across the disc.

To highlight the difference between traditional center-based camera motion and surface-based camera motion, see the example of motion about a cylinder in Figure 4. With the simple traditional tumble, the rotation about the cylinder would have placed the camera inside the object. However, with HoverCam, moving to the right pans the camera until it can rotate about to continue panning across the end disc.

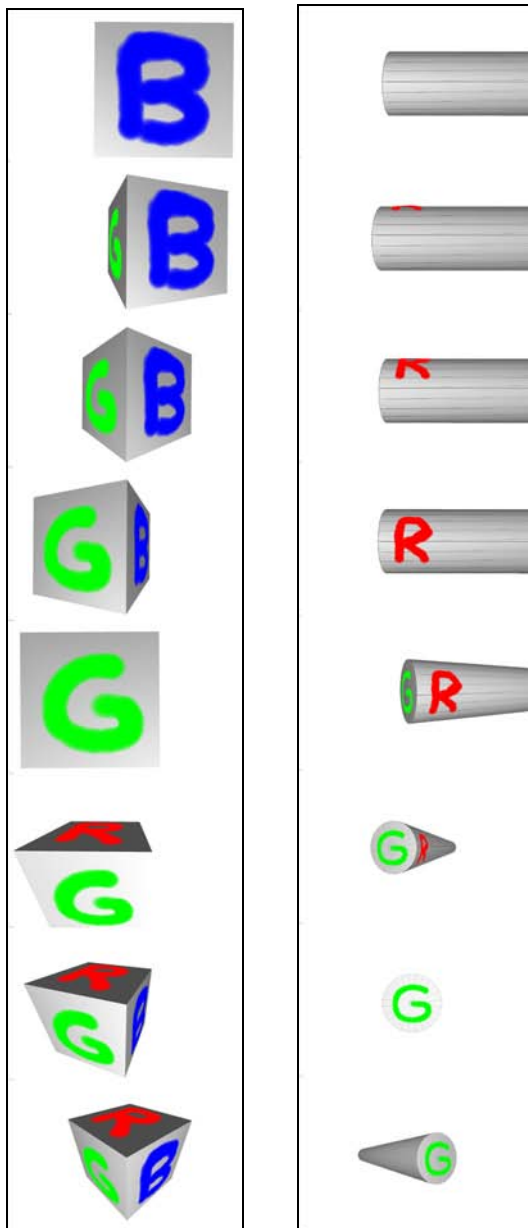


Figure 3. HoverCam around a cube and a cylinder, from the point of view of the user.

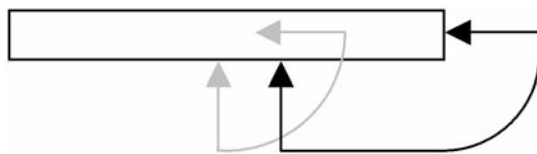


Figure 4. Simple Rotation versus HoverCam: The grey path shows how a simple rotation about the center of a cylinder leaves the camera within the object. However, HoverCam moves the camera along the cylinder and only rotates when turning about the end of the shaft (black path).

2.2 Blending Camera Techniques

As HoverCam would normally be used together with traditional freeform navigation tools, we have designed HoverCam to interoperate between the various camera techniques in a fairly seamless way. Freeform camera motion allows the user to navigate to any point in space and to face in any direction. For specific surface-based tasks like 3D painting or sculpting, HoverCam provides a subset of this freedom with the benefit of following the surface. Switching from HoverCam to a freeform camera could simply be invoked by clicking on a tool icon or by a key press. However, switching from a freeform camera to HoverCam may cause an abrupt reorientation and reposition of the camera because an initial search may find a result quite far from the current view. Two methods are used to ease this disruption. In the case where a freeform camera approaches an object from a significant distance, a field of influence around the object specifies how strongly the HoverCam motion is linearly interpolated with the freeform motion. In the case where a freeform camera is already very close to an object (fully within the HoverCam field), motion clipping (as discussed in Section 2.5) is applied to smoothly transition to HoverCam motion.

Layers of HoverCam influence around each object are automatically generated (see Figure 5). The outer layer is quite far from the surface and specifies a *field of influence*. Once a freeform camera enters this field, the HoverCam camera is weighted together with the freeform camera so that it will be sucked towards the outer limit of the orbit distance (see Figure 6). Once the camera is fully controlled by the HoverCam algorithm, it remains so until the user switches to another navigation method. In practice, we have found it helpful for the user to be able to specify the distance between the surface and the camera. In our current implementation, the mouse-wheel is used to zoom in or out to specify a new fixed distance to the HoverCam algorithm. The HoverCam camera orbit distance will always be between the inner limit and the outer limit unless the user zooms out beyond the field of influence.

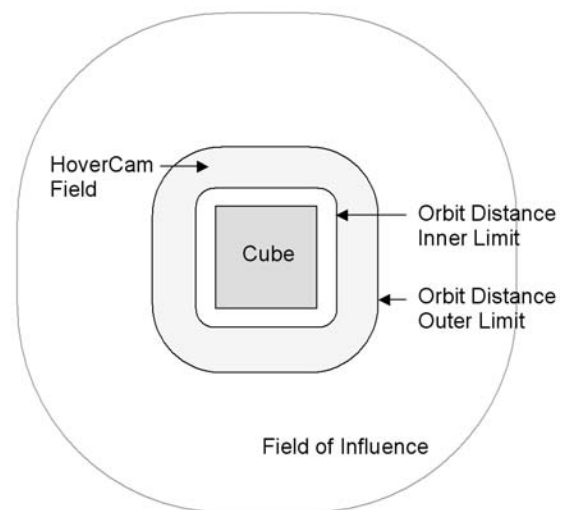


Figure 5. HoverCam Layers: A large outer shell acts a type of gravity field that interpolates traditional camera motion with the HoverCam camera motion until the Outer Limit of the Orbit Distance is reached.

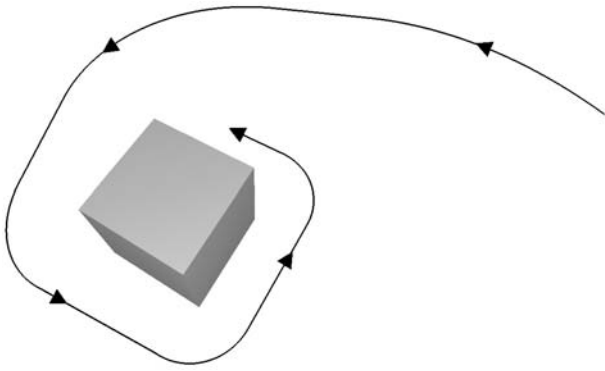


Figure 6. HoverCam Layers: As the camera approaches an object, HoverCam is slowly engaged.

2.3 Different Notions of “Up”

When closely inspecting an object in an abstract empty virtual environment, the problem of correctly orienting the camera, so objects do not appear to be sideways or upside-down, is not trivial. Furthermore, the model chosen to derive the up-vector at a given camera position, or given a certain camera motion, may alter the overall camera metaphor. We define four up-models: Global, Local, Driving, and Custom.

Global: Consider a globe representing the earth. Regardless of where the camera may be positioned, or how it moves, *up* is typically the direction toward the North Pole. For example, whether the user is looking at Australia or Sweden, the camera would be oriented so that the North Pole would be toward the top of the screen. If the user moved across the North Pole from Canada to Russia, the camera would effectively spin about 180° so that it would come down on the Russian side, but with the North Pole still toward the top of the screen. This constant up-vector high above the center of the scene defines our Global Up-Vector Model as shown in Figure 7.

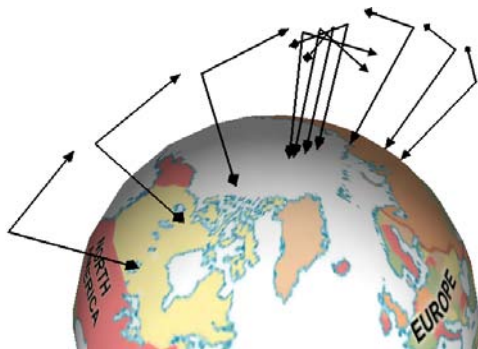


Figure 7. Global Up-Vector Model.

Local: In this egocentric model, the up-vector is view dependent and always points toward the top of the viewport. Therefore, moving the cursor left or right does not affect the up-vector. However, moving up or down causes the up-vector to be corrected so that the user never feels as though they have turned. For example, when moving over the North Pole of a globe from Canada to Russia, if Canada initially looked the right way up, Russia would appear upside-down. See Figure 8.

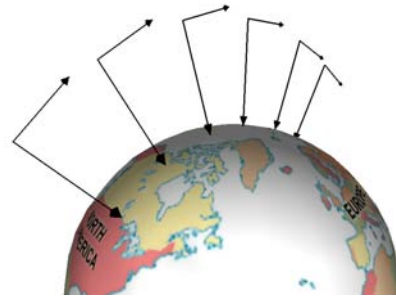


Figure 8. Local Up-Vector Model.

Driving: For some objects, the user may wish to have the feeling that moving the input device left or right should turn the object so that moving (the device) *up* is always “forward”. Again, using the globe as an example, if we started over Brazil with the equator horizontal across the view and we moved the input device to the right, the horizon would rotate in the view until vertical, with the North Pole toward the left hand side of the screen. See Figure 9. This model could also be considered for a “flying” camera metaphor since it smoothly banks the camera in the left or right direction of mouse motion.

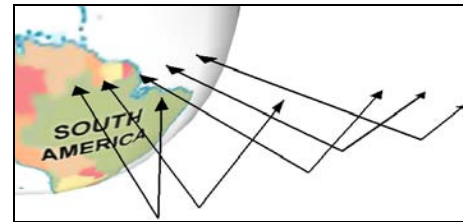


Figure 9. Driving Up-Vector Model.

Custom: Finally, some objects may require custom up-vector fields. For example, a model of an automobile would normally have the up-vector point from the car to high above the top of the roof. However, if a user was looking underneath the car or above the car, it may seem proper to have *up* be towards the hood. In this case, custom up-vectors could be placed on the sides of an enclosing cube, which would be interpolated based on the current camera position, to determine the current up-vector. In our current implementation, the user can move to any point in space and press a hotkey to generate an up-vector at the current position and orientation. In this way, a complex up-vector field may be authored. See Figure 10.

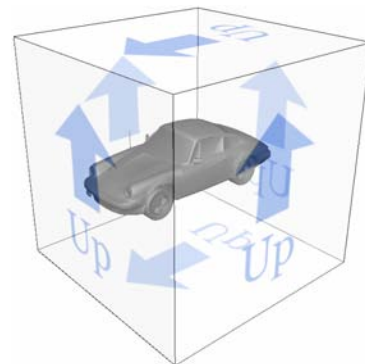


Figure 10. Custom Up-Vector Model.

2.4 Input Mapping

The mapping of mouse motion to camera motion may either be *push* (egocentric) or *pull* (exocentric). In a push mapping, the user conceptually pushes the camera so that, for example, moving the mouse from left to right would push the camera to the right causing the model to appear to move left. With a pull mapping, the user pulls the world in the direction of motion so that dragging from left to right moves the object from left to right, achieved by moving the camera to the left.

The name of our technique –HoverCam– implies that the user is controlling a craft floating above the surface of an object and so one may expect the push mapping to be most effective. However, given our object inspection metaphor, we typically expect to be very close to the object so that it fills most of the display. As such, when the user clicks to drag the mouse, the cursor will typically be over the object when the mouse button is clicked. This strongly conveys a metaphor of grabbing the object at that position and dragging it in the mouse direction, which implies that the camera will move in the opposite direction. For this reason, we chose the pull mapping.

Still, during a single click-drag-release input event series, a discrepancy can occur between the direction that the input device is moving and the intended camera motion in the scene. For example, for the camera motion shown in Figure 7, the user would move the mouse down until they reached the North Pole, but continuing to move down would do nothing. To move down the other side, the user would have to move the mouse in an upward direction. This can make the user feel as though they are “stuck” and this can be fairly confusing. To fix this discrepancy, HoverCam uses two up-models: one for internal calculations and one for display to the user. Internally, the Local up-model is used, which will move continuously across the top of the globe in a single drag motion without getting stuck, as shown in Figure 8. However, the *up* effect that the user sees may be any one of the four methods described above. This is implemented by applying the Local model to the camera position and orientation, followed by the application of the chosen up-model. As this update is applied during every mouse-move event, the user only feels the effect of the chosen up-model.

An appropriate choice for the up-vector model may be highly content dependent and may be a user preference or may be uniquely associated with each model in a scene.

2.5 Fighting Cavities

This basic algorithm shown in Figure 2 nicely handles simple convex surfaces, slightly concave surfaces, and jumps across gaps or holes. However, the true closest point may be outside the current field of view (FOV) or may even be behind the camera. In these cases, turning the camera to immediately face the new closest point would be quite disorienting and may result in some undesirable effects. This can occur if the object has protrusions, or cavities. When gliding over a cavity, for example, the closest point will jump from one edge of the cavity to the other. Step (e) of the algorithm clips the final distance traveled (of both the eye and the look-at point) to minimize these effects, slowly turning the camera to the intended position.

Specifically, to maintain smooth camera motion, Step (e) looks at the vectors (see Figure 11) from the old closest point to the new closest point (L_0L_2) and from the old eye position to the new eye position (E_0E_2). We then clip these vectors to the length

δ of the input vector i generated by the mouse move. This creates the final *eye to look-at* vector E_3L_3 .

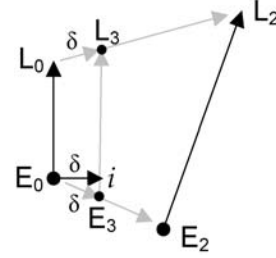


Figure 11. Motion Clipping. The final position of E_0 and L_0 are clipped from E_2 and L_2 to E_3 and L_3 .

This motion-clipping step handles sharp camera turns and jumps across holes in an object or jumps across gaps to other objects. Figure 12 shows the HoverCam camera path while moving across the top of a torus (from left to right). Note how extra steps are generated across the hole in the torus, when the closest point is on the right-hand side, to smoothly turn toward the other side of the torus to continue around it.

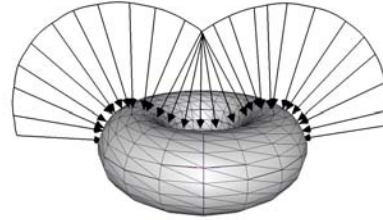


Figure 12. When moving across the hole in the torus (from left to right), HoverCam generates the extra steps needed to maintain smooth motion.

However, the camera motion shown in Figure 12 is only possible with an additional constraint. At the point where the camera is directly over the center of the torus, there are an infinite number of solutions when searching for the closest point. To resolve cases such as this, and to favor the user input, a restricted FOV constraint is added in step (b) to only look for the closest point in the general direction of the input vector (see Figure 13). There are two inputs to this constraint: the input vector i and the angle of the field of view β . In our current implementation, β is fixed at 45° , but could be based on the view frustum. P_0 is the point along the vector formed by adding i to L_0 and ensuring $L_0E_0P_0$ is $\frac{1}{2}\beta$. All geometry outside the triangular wedge determined by $E_0L_0P_0$, with a thickness of 2δ and a length extending infinitely away from E_0 , is disregarded during the search for the closest point to E_0 .

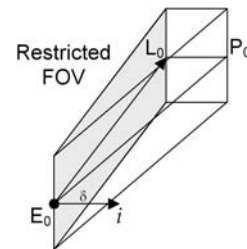


Figure 13. Restricted Search FOV. By restricting the search volume for a new closest point, the camera motion favors the user input.

This constraint helps HoverCam to handle a number of situations. In the torus example, a new closest point is found directly across the hole and the motion clipping turns the camera toward it. In concave shapes, where again, an infinite number of points are all equally close to the eye position, the user input helps to uniquely select a subset of results (see Figure 14).

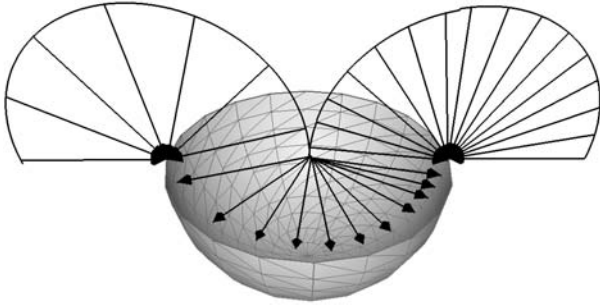


Figure 14. When moving across the inside of an open sphere (from left to right), the Restricted Search FOV and the motion clipping work together to create the expected camera motion.

To summarize, the restricted FOV for searching (in step (b)) taken together with the motion clipping (step (e)), handle the cases where there are multiple solutions thereby providing the expected camera motion.

2.6 Handling Sharp Turns

While the basic camera update steps outlined above generate smooth camera motion paths, tight corners can create hooks in the path that could be avoided. The problem is caused by the restricted search FOV that prevents the algorithm from finding an upcoming corner. For example, when moving right along a wall towards a corner, HoverCam looks directly ahead at the wall while panning right. However, the restricted FOV prevents HoverCam from seeing the approaching corner. The corner will eventually be found but this will push the camera back to the fixed distance from the surface effectively generating a hook in the camera path (see Figure 15).

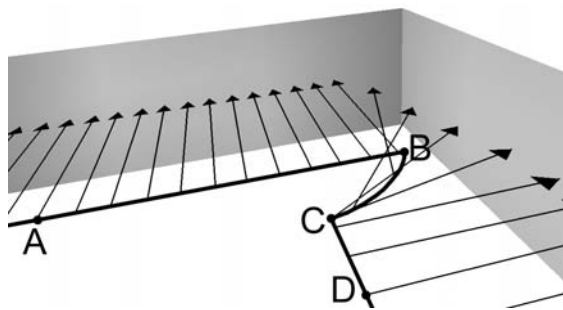


Figure 15. Hook in camera motion path when turning in a corner while moving left to right (from A to D).

To achieve the preferred trajectory, HoverCam includes a second FOV that searches for obstacles in the direction of motion (see Figure 16). The search for the closest point then includes both the restricted search FOV and the obstacle FOV. The closest point in either FOV will be considered to be the target point that we would like to veer towards.

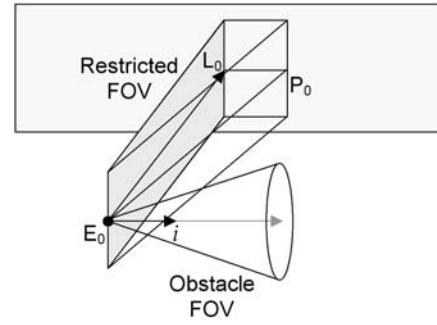


Figure 16. Restricted FOV along underlying surface and Obstacle FOV looking ahead in the direction of movement, as specified by the input vector i .

Now, when a corner is reached, the camera correctly turns in the direction of the input until it continues along the next wall (see Figure 17). The imminent collision with the wall is detected and the closest point will then be contained on that wall. Several steps are made while the camera turns toward it after which the camera carries on normally.

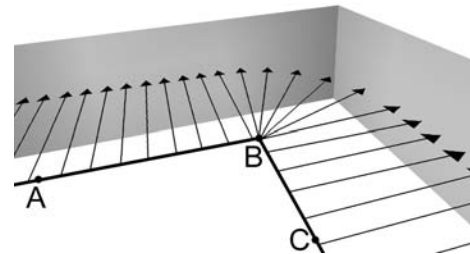


Figure 17. No hook in camera motion path when turning in a corner, while moving left to right (from A to C).

3 Implementation

We implemented a HoverCam prototype application in C++ under Windows XP. We added basic functionality for loading Wavefront (obj) models and rendering them using of the OpenGL graphics library. We also added visualization functions to record user input and draw the motion paths shown in the figures in this paper.

As outlined above, the general HoverCam algorithm is based on a closest point search across a polygon mesh. For obvious reasons, the naïve approach of iterating through every polygon in the model for closest point analysis would be too costly to be used for an interactive operation such as HoverCam. We therefore generate an indexing structure called a sphere-tree when the user loads an object. The sphere-tree is a hierarchal structure that encloses the polygons within our model (see Figure 18) and is built using a modified octree algorithm.

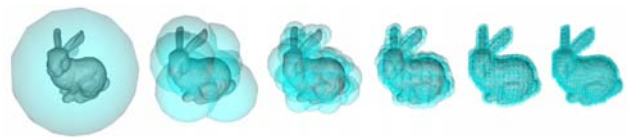


Figure 18. Six successive levels of a sphere-tree enclosing a 3D bunny model. Notice how closely the sphere-tree represents the model at the low levels.

To compute an approximate closest point on the surface of the mesh, we perform a top-down traversal of our sphere-tree, maintaining a list of all the spheres that satisfy our FOV constraints. If a sphere fails to be within either of our FOVs, we eliminate it from the list without exploring any of its children. In this manner, we eliminate a large majority of our polygons from the closest point analysis. As a further optimization, we also find the distance from the query point to the far end of each sphere, and eliminate all the spheres further than the current minimum distance. The traversal terminates when the bottom of the sphere-tree is reached, and the result is a list of the smallest spheres from the lowest level. The polygons contained within these spheres are then subjected to regular closest point analysis.

It is often the case that some of the polygons lie on the boundary of our restricted FOV. For these polygons, if the true closest point is outside the FOV, then the closest point we are interested in will lie along the intersection of our FOV with the polygon. When this happens, we perform a ray-casting technique about the perimeter of the FOV, finding the closest intersection of a ray with the polygon. If our search method ignored these cases, HoverCam would mistakenly only select closest points on polygons completely within the FOV.

4 Limitations

Our HoverCam algorithm essentially handles all static 3D models. The model can have significant protrusions and cavities (convex and concave areas) and may even be interior spaces such as a game level. However, moving objects may not always be handled properly, especially if moving faster than the camera. Also, models with very fine protrusions around the camera may not be found if they fall between the two FOVs being used. Any of these conditions may cause HoverCam to move inside the object or to miss it entirely.

Another limitation exists in the closest point search method. If a fast search method cannot be provided to HoverCam, interactive rates will suffer. For example, the automobile model in Figure 10 has 21,000 polygons unevenly distributed in space. Due to the high concentration of thousands of polygons in the wheels of the car, gliding across the wheels noticeably slows camera movement, despite a fairly efficient sphere tree implementation.

5 Initial Impressions

We showed HoverCam to six target users who were advanced 3D modelers and animators to get their initial impression. After describing the basic interaction model, we asked them to use HoverCam to inspect one of our 3D car models. All of them understood the concept and interaction mechanisms and could easily inspect the car. The camera orientation (including up-model) worked exceptionally well. In addition, a few of the users opted to use both the HoverCam and at times the traditional camera controls to inspect the car. Our system seamlessly blended the two camera styles. Finally, one user commented that HoverCam is "better than shifting between individual modes."

The most distracting usability issue appears to be the "shakiness" of the HoverCam technique as many users commented on the problem. This is an artifact of following faceted surfaces too closely. We can easily address this by smoothing normals or smoothing the model mesh. For future

work, we may add level of detail support so that when HoverCam is further from the object, a smoother version of the model can be used to control the camera. In addition, two of the users requested the ability to get to an exactly framed spot (e.g., a close-up of a side mirror). The orbit distance inner limit must be small enough to allow for these types of close-up shots as we learned that our initial inner limit distance was too large. In the end, all of the users saw the value of HoverCam.

6 Related Work

A great deal of prior research has explored camera techniques for 3D virtual environments. Many of the techniques use 2D input from a mouse or stylus and introduce metaphors to assist the user. The most pervasive metaphor is the cinematic camera model, enabling users to rotate, pan and zoom the viewpoint. Researchers have also explored other camera metaphors including orbiting and flying [Tan et al. 2001], using constraints [Mackinlay et al. 1990; Smith et al. 2001], drawing a path [Igarashi et al. 1998], through-the-lens control [Gliecher and Witkin 1992], points and areas of interests [Jul and Furnas 1998], two-handed techniques [Balakrishnan and Kurtenbach 1999; Zeleznik et al. 1997], and combinations of techniques [Steed 1997; Zeleznik and Forsberg 1999]. Bowman et. al. present taxonomies and evaluations of various interactions and camera models [1997; 1999].

Systems that utilize higher degree-of-freedom input devices offer additional control and alternative metaphors have been investigated, including flying [Chapman and Ware 1992; Ware and Fleet 1997], eyeball-in-hand [Ware and Osborne 1990], and worlds in miniature [Stoakley et al. 1995]. Other techniques involve automatic framing of the areas of interest as typically found in game console based adventure games which use a "chase airplane" metaphor for a third person perspective. Rules can also be defined, for cameras to automatically frame a scene, that follow cinematic principles such as keeping the virtual actors visible in the scene; or following the lead actor [He et al. 1996]. Researchers have also investigated so-called *guided tours* where camera paths are procedurally determined or pre-specified for the end user to travel along. Galyean [1995] proposes a "river analogy" where a user, on a metaphorical boat, can deviate somewhat from the river, by steering using a conceptual "rudder". Hanson and Wernert [1997; 1999] propose "virtual sidewalks" which combine virtual surfaces and specific gaze direction, and vistas along the sidewalk. Wan et al. determine a best path for automatic fly-through medical applications [2001].

The most directly related work is the UniCam [Zeleznik and Forsberg 1999] click-to-focus feature and the Tan et al. [2001] navigation system. Both of these systems are suites of camera manipulation tools and both have one feature that examines the in-scene geometry. Once the user has clicked on an object of interest, a camera path is generated to move and orient the camera toward the selected target point. The UniCam system animates the view to the new position while the Tan system uses keyboard keys to move along the generated path.

Our technique differs from these in that an updated position and orientation is interactively generated so the user is continuously in control of the camera motion and can change directions at any time. Also, the two systems mentioned do not perform collision detection or obstruction detection and so, may pass through

other polygons. Finally, HoverCam handles convex and concave shapes and models an up-vector field, whereas neither of the other systems address these aspects of navigation.

7 Other Applications: Volumetric Operations

In addition to surface based navigation, HoverCam can be used to intelligently view geometry without well-defined surfaces such as curves in space, point cloud data sets, or volumetric densities. To support navigation about lines or points, only the closest point search function must be changed. Figure 19 shows a HoverCam camera path made by a user moving around a set of randomly generated points (drawn as small spheres). The displayed camera path shows that HoverCam keeps the cloud data as the center of interest as the user moves around the cloud from right to left.

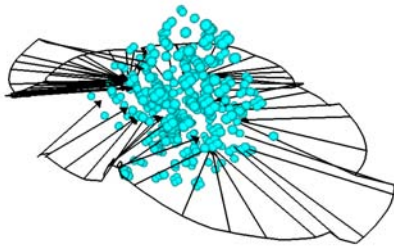


Figure 19. HoverCam Navigation about a Point Cloud.

The HoverCam algorithm can also be used to create volumetric densities. With an additional button, HoverCam can perform other operations such as selection or painting. Figure 20 shows a curve in space around which HoverCam can travel. When the user presses a modifier key, HoverCam leaves a paint trail as it moves about the curve. By increasing or decreasing the orbit distance, the user can paint closer or further from the base curve.

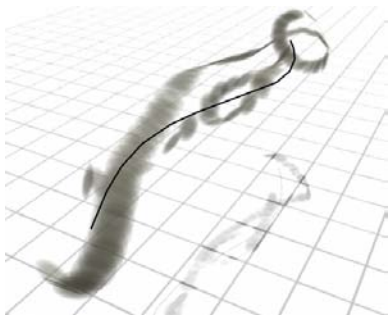


Figure 20. Painting volumetric density with HoverCam.

8 Conclusion

In this paper we introduced a new technique for interactive object inspection called HoverCam. The fundamental principle is to move the camera, under a small set of constraints including collision detection in the hover direction and the motion direction, followed by a small number of corrections, to maintain the hover distance from the object.

There are a number of applications of this algorithm including object inspection, volumetric operations, and interior navigation. The primary benefit to users is a simplified interaction that only requires 2D input, which can be engaged with just one button or control. Also, for object inspection, novice users can move

around an object without moving to awkward positions or orientations.

Acknowledgments

The authors thank the participants of our user study, and Don Almeida for early implementation work.

References

- BALAKRISHNAN, R. and KURTENBACH, G. 1999. Exploring bimanual camera control and object manipulation in 3D graphics interfaces. *ACM CHI*. 56-63.
- BOWMAN, D., JOHNSON, D. and HODGES, L. 1997. Travel in immersive virtual environments. *IEEE VRAIS*. 45-52.
- BOWMAN, D., JOHNSON, D. and HODGES, L. 1999. Testbed environment of virtual environment interaction. *ACM VRST*. 26-33.
- CHAPMAN, D. and WARE, C. 1992. Manipulating the future: predictor based feedback for velocity control in virtual environment navigation. *ACM Symposium on Interactive 3D Graphics*. 63-66.
- GALYEAN, T. 1995. Guided navigation of virtual environments. *ACM Symposium on Interactive 3D Graphics*. 103-104.
- GLIECHER, M. and WITKIN, A. 1992. Through-the-lens camera control. *ACM SIGGRAPH 92*. 331-340.
- HANSON, A. and WERNET, E. 1997. Constrained 3D navigation with 2D controllers. *IEEE Visualization*. 175-182.
- HE, L., COHEN, M. and SALESIN, D. 1996. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. *ACM SIGGRAPH 96*. 217-224.
- IGARASHI, T., KADOBAYASHI, R., MASE, K. and TANAKA, H. 1998. Path drawing for 3D walkthrough. *ACM UIST*. 173-174.
- JUL, S. and FURNAS, G. 1998. Critical zones in desert fog: aids to multiscale navigation. *ACM UIST*. 97-106.
- MACKINLAY, J., CARD, S. and ROBERTSON, G. 1990. Rapid controlled movement through a virtual 3D workspace. *ACM SIGGRAPH 90*. 171-176.
- SMITH, G., SALZMAN, T. and STUERZLINGER, W. 2001. 3D Scene manipulation with 2D devices and constraints. *Graphics Interface*. 135-142.
- STEED, A. 1997. Efficient navigation around complex virtual environments. *ACM VRST*. 173-180.
- STOAKLEY, R., CONWAY, M. and PAUSCH, R. 1995. Virtual reality on a WIM: Interactive worlds in miniature. *ACM CHI*. 265-272.
- TAN, D., ROBERTSON, G. and CZERWINSKI, M. 2001. Exploring 3D navigation: combining speed-coupled flying with orbiting. *ACM CHI*. 418-425.
- WAN, M., DACHILLE, F. and KAUFMAN, A. 2001. Distance-Field Based Skeletons for Virtual Navigation. *IEEE Visualization 2001*. 239-245.
- WARE, C. and FLEET, D. 1997. Context sensitive flying interface. *ACM Symposium on Interactive 3D Graphics*. 127-130.
- WARE, C. and OSBORNE, S. 1990. Exploration and virtual camera control in virtual three dimensional environments. *ACM Symposium on Interactive 3D Graphics*. 175-183.
- WERNERT, E. and HANSON, A. 1999. A framework for assisted exploration with collaboration. *IEEE Visualization*. 241-248.
- ZELEZNIK, R. and FORSBERG, A. 1999. UniCam - 2D Gestural Camera Controls for 3D Environments. *ACM Symposium on Interactive 3D Graphics*. 169-173.
- ZELEZNIK, R., FORSBERG, A. and STRAUSS, P. 1997. Two pointer input for 3D interaction. *ACM Symposium on Interactive 3D Graphics*. 115-120.

4.4 Paper 4: [GW92]

Michael Gleicher and Andrew Witkin *Through-the-lens camera control*. In Proceedings of the Siggraph Conference (1992), ACM Computer Graphics, pp. 331-340.

Through-the-Lens Camera Control

Michael Gleicher and Andrew Witkin
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

{gleicher|witkin}@cs.cmu.edu

Abstract

In this paper we introduce *through-the-lens camera control*, a body of techniques that permit a user to manipulate a virtual camera by controlling and constraining features in the image seen through its lens. Rather than solving for camera parameters directly, constrained optimization is used to compute their time derivatives based on desired changes in user-defined controls. This effectively permits new controls to be defined independent of the underlying parameterization. The controls can also serve as constraints, maintaining their values as others are changed. We describe the techniques in general and work through a detailed example of a specific camera model. Our implementation demonstrates a gallery of useful controls and constraints and provides some examples of how these may be used in composing images and animations.

Keywords: camera control, constrained optimization, interaction techniques

1 Introduction

Camera placement and control play an important role in image composition and computer animation. Consequently, considerable effort has been devoted to the development of computer graphics camera models. Most camera formulations are built on a common underlying model for perspective projection under which any 3-D view is fully specified by giving the center of projection, the view plane, and the clipping volume. Within this framework, camera models differ in the way the view specification is parameterized. Not all formulations are equivalent—some allow arbitrary viewing geometries, while others impose restrictions. Even so, alternative models can be viewed to

a great degree as alternative slicings of the same projective pie.

How important is the choice of the camera model's parameterization? Very important, if the parameters are to serve directly as the controls for interaction and keyframe interpolation. For example, the popular LOOKAT-/LOOKFROM/VUP parameterization makes it easy to hold a world-space point centered in the image as the camera moves without tilting. To do the same by manually controlling generic translation/rotation parameters would be all but hopeless in practice, although possible in principle.

The difficulty with using camera parameters directly as controls is that no single parameterization can be expected to serve all needs. For example, sometimes it is more convenient to express camera orientation in terms of azimuth, elevation and tilt, or in terms of a direction vector. These particular alternatives are common enough to be standardly available, but others are not. A good example involves the problem, addressed by Jim Blinn[3] of portraying a spacecraft flying by a planet. Blinn derives several special-purpose transformations that allow the image-space positions of the spacecraft and planet to be specified and solves for the camera position. The need for this kind of specialized control arises frequently, but we would rather not face the prospect of deriving and coding specialized transformations each time they do.

In short, camera models are inflexible. To change the controls, one must either select a different pre-existing model or derive and implement a new one. If this inflexibility could be removed, the effort devoted to camera control could be reduced and the quality of the result enhanced.

In this paper, we present a body of techniques, which we call *through-the-lens camera control*, that offer a general solution to this problem. Instead of a fixed set, the user is given a palette of interactive image-space and world-space controls that can be applied "on the fly," in any combination. For example, the image-space position of an arbitrary world-space point can be controlled by interactive dragging, or pinned while other points are moved. Image-space distances, sizes, and directions can also be

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

controlled. Points can be constrained to remain within the image or within a specified sub-region. These and other image-space controls can be freely combined with direct world-space controls on camera position and orientation. The set of controls is extensible, with a general procedure for adding new ones.

Using through-the-lens control, the spacecraft/planet problem, and others of its kind, could be solved immediately and interactively: a point on the planet would be grabbed, dragged to its target location, then left pinned at that image point. The spacecraft would be similarly positioned and pinned. Residual degrees of freedom could be fixed by dragging additional points, or manipulating the camera in world space, while both image points remained nailed. All the while, the required camera motions would be computed automatically at interactive speed.

The principal technical obstacle to achieving this kind of control lies in the nonlinearity of the relationship between the desired controls and the underlying view specification. No general guaranteed procedure exists for solving nonlinear algebraic systems; in fact there may often exist no solution, or many. The direct approach—solving numerically for the camera parameters given the controls—is therefore unlikely to succeed.

The key to our approach is that we instead formulate the problem differentially—solving for the *time derivatives* of the camera parameters, given the *time derivatives* of the controls. For example, a point is dragged by specifying its velocity from moment to moment, rather than giving a final target position. When the camera is under interactive control, it falls to the user interface to convert user actions, such as pointer motions, into suitable velocity signals. In keyframe control, the velocity is calculated by taking the time derivative of the interpolating function. The use of differential control does not allow us to directly position the camera in global leaps, but instead provides a robust and accurate means of translating continuous adjustments of the controls into continuous motions of the camera. We are primarily interested in interactive control by grabbing and dragging, for which continuous motion is desirable, and with low-level camera control for animation, for which continuous motion is sufficient.

We formulate the problem of computing time derivatives of camera parameters as a simple constrained optimization. Once the derivatives have been computed, using them to update the camera's state over time reduces to the standard problem of solving a first-order ordinary differential equation from an initial value, for which good numerical methods abound (see Press *et. al.* [20] for a good practical introduction.)

An interesting feature of through-the-lens control is the new role in which it places the camera parameterization. Although we retain a fixed underlying camera model, the model's parameters no longer bear the burden of serving as user-level controls. In fact, the parameters may be com-

pletely hidden from the user. This leaves us free to choose a camera model on the basis of numerical well-behavedness and implementation convenience. Our preferred formulation, based on quaternion rotations, is a case in point: it is a very poor model by conventional criteria, since the four components of the quaternion would be exceedingly difficult to control directly. Yet it provides an ideal substrate for through-the-lens control because it allows free camera rotations without singularities or other artifacts. We avoid the well-known difficulties involved with interpolating quaternions for animation[22] by interpolating the controls instead.

The remainder of the paper is organized as follows: Following a discussion of related work, we develop the machinery of through-the-lens control in terms of a generic camera model, starting with the problem of controlling the image-space velocity of a single point, then generalizing to the full solution. We then present complete through-the-lens equations for our simple quaternion-based camera model. We describe our implementation and present examples, then conclude with a discussion of future work.

2 Related Work

As we noted in Section 1, standard computer graphics camera models are based on specialized transformations that specify the view as a function of parameters that are useful for interactive, procedural, or keyframed control. Earlier we discussed the standard LOOKAT/LOOKFROM model. An example of a more general viewing model currently in wide use is the PHIGS+ model[6]. In addition, a variety of special-purpose models such as Blinn's spacecraft flyby transformations [3] have been developed. Issues involved in using the LOOKAT/LOOKFROM model to navigate virtual spaces are considered by [16]. In [7], the LOOKAT/LOOKFROM model is embedded in a procedural language for specifying camera motions.

Much of the work on interactive camera placement in computer graphics has been concerned with direct control of the camera's position and orientation. The problem of developing intuitive controls for 3-D rotations is a difficult one[5], particularly when the input device is two-dimensional. Several researchers have addressed the problem through the use of 3-D interfaces, including six degree-of-freedom pointing devices[26, 25, 1] and more specialized devices such as steerable treadmills[4].

Problems involving the recovery of camera parameters from image measurements have been addressed in photogrammetry¹, computer vision, and robotics. All of these are concerned with the recovery of parameter values, rather than time derivatives. Algebraic solutions to specific problems of this kind are given in [18] and [9], while numerical solutions are discussed in [15, 10, 17]. In [24], constrained optimization is employed to position a

¹Also see chapter 6 of [21] for amazing mechanical solutions to photogrammetry problems.

real camera, mounted on a robot arm, for the purpose of object recognition. Factors considered in the optimization include depth of field, occlusion, and image resolution.

Optimization techniques have been applied to the related problem of object placement in computer graphics. In [1] articulated figures are posed using penalty methods to meet positional goals. In [28], similar methods are employed for general object placement and control. The use of these methods for camera placement in animation is described in [30].

The differential control methods employed in this paper are formally more closely allied to the methods of constrained dynamic simulation described in [29, 2, 19, 31, 23] than to the positional optimization methods cited above. Some of the issues involved in adapting these methods to differential kinematic control are addressed in [13], while [27] considers their application to the design of free form surfaces, and [12] illustrates their use in a constraint-based drawing program.

3 The Machinery

In this section we introduce the basic mechanisms that support through-the-lens control, employing a simple constrained optimization formulation. Assuming that we have chosen a camera model to provide the fixed, underlying parameterization, we solve for the time derivatives of the parameters such that their mean squared deviation from a desired value is minimized, subject to the constraints imposed by the image-space controls. Setting the default values to zero yields a solution that minimizes the mean squared rate of change of camera parameters. Non-zero values can be used to support interactive dragging subject to the constraints imposed by other controls.

We begin by giving the relationship between a world-space point and its image-space counterpart, which we express in terms of a generic camera model. A specific quaternion-based model will be fully described in section 4. We give the coordinates of an image point \mathbf{p} as

$$\mathbf{p} = \mathbf{h}(\mathbf{V}\mathbf{x}), \quad (1)$$

where \mathbf{x} is the world-space point that projects to \mathbf{p} , \mathbf{V} is a homogeneous matrix representing the combined projection and viewing transformations, and \mathbf{h} is a function that converts homogeneous coordinates into 2-D image coordinates, defined by

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 \\ x_4 & x_4 \end{bmatrix},$$

where the x_i 's are components of homogeneous point \mathbf{x} . The matrix \mathbf{V} is some (for now unspecified) function of the camera model parameters, which we denote by a length- n vector \mathbf{q} . In practice, \mathbf{V} would usually be computed as the product of several matrices, each a function of one or more of the parameters.

3.1 Camera motions and image point velocities.

Assuming for now that the world space point \mathbf{x} is fixed, the image point \mathbf{p} is entirely a function of the camera parameters \mathbf{q} . This is a nonlinear relationship because \mathbf{h} is nonlinear, as in general is $\mathbf{V}(\mathbf{q})$. We obtain the expression for the image velocity $\dot{\mathbf{p}}$ by applying the chain rule:

$$\dot{\mathbf{p}} = \mathbf{h}'(\mathbf{V}\mathbf{x}) \left(\frac{\partial(\mathbf{V}\mathbf{x})}{\partial \mathbf{q}} \right) \dot{\mathbf{q}}, \quad (2)$$

where $\mathbf{h}'(\mathbf{x})$ is the matrix representing the derivative of $\mathbf{h}(\mathbf{x})$, given by

$$\mathbf{h}'(\mathbf{x}) = \begin{bmatrix} \frac{1}{x_4} & 0 \\ 0 & \frac{1}{x_4} \\ 0 & 0 \\ -\frac{x_1}{x_4^2} & -\frac{x_2}{x_4^2} \end{bmatrix}. \quad (3)$$

$\dot{\mathbf{q}}$ is the time derivative of \mathbf{q} , and $\partial(\mathbf{V}\mathbf{x})/\partial \mathbf{q}$ is the $4 \times n$ matrix representing the derivative of the transformed point $\mathbf{V}\mathbf{x}$ with respect to \mathbf{q} . We differentiate the point $\mathbf{V}\mathbf{x}$ rather than the matrix \mathbf{V} to avoid differentiating a matrix with respect to a vector, which would give rise to a rank-3 tensor. In section 4, we give an example of how this derivative matrix can be computed.

For notational compactness we will define the $2 \times n$ matrix

$$\mathbf{J} = \mathbf{h}'(\mathbf{V}\mathbf{x}) \frac{\partial(\mathbf{V}\mathbf{x})}{\partial \mathbf{q}}, \quad (4)$$

so that

$$\dot{\mathbf{p}} = \mathbf{J}\dot{\mathbf{q}}. \quad (5)$$

Notice that equation 5 gives $\dot{\mathbf{p}}$ as a *linear* function of $\dot{\mathbf{q}}$, even though \mathbf{p} is a nonlinear function of \mathbf{q} .

3.2 Controlling a single point

Having obtained $\dot{\mathbf{p}}$ as a function of $\dot{\mathbf{q}}$, we next consider the problem of controlling a single image point, i.e. solving for a value of $\dot{\mathbf{q}}$ that makes the image point assume a given velocity $\dot{\mathbf{p}} = \dot{\mathbf{p}}_0$. In practice the value for $\dot{\mathbf{p}}_0$ might be supplied by the user interface or might indicate the velocity on a keyframed motion path. Although the relation between $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ is linear, we cannot simply solve $\dot{\mathbf{p}}_0 = \mathbf{J}\dot{\mathbf{q}}$ for $\dot{\mathbf{q}}$ unless matrix \mathbf{J} is square and of full rank, which in general it will not be.

The singularity of the matrix \mathbf{J} reflects the fact that many distinct camera motions can cause a single point to move in the same way. One way to solve the problem might be to require the user to control enough points or other features to yield a square matrix. We choose a different option that offers far more flexibility: subject to the constraint that $\dot{\mathbf{p}} = \dot{\mathbf{p}}_0$, we minimize the magnitude of $\dot{\mathbf{q}}$'s deviation from a specified value $\dot{\mathbf{q}}_0$. Letting $\dot{\mathbf{q}}_0 = 0$ imposes a criterion of minimal change in the camera parameters. As we shall see later, the ability to choose other values makes it possible, for example, to drag image points and other features *subject to* the hard constraints.

The problem we now wish to solve is:

$$\text{minimize } E = \frac{(\dot{\mathbf{q}} - \dot{\mathbf{q}}_0) \cdot (\dot{\mathbf{q}} - \dot{\mathbf{q}}_0)}{2} \text{ subject to } \dot{\mathbf{p}} - \dot{\mathbf{p}}_0 = 0. \quad (6)$$

To qualify as a constrained minimum, $\dot{\mathbf{q}}$ must satisfy several conditions. First, of course, the constraint must be met, i.e.

$$\dot{\mathbf{p}}_0 = \mathbf{J}\dot{\mathbf{q}}.$$

At an unconstrained minimum, we would require that the gradient $dE/d\dot{\mathbf{q}}$ vanish. Instead, we require that it point in a direction in which displacements are prohibited by the constraints. This condition is expressed by requiring that

$$dE/d\dot{\mathbf{q}} = \dot{\mathbf{q}} - \dot{\mathbf{q}}_0 = \mathbf{J}^T \lambda,$$

for some value of the 2-vector λ of *Lagrange multipliers*. This equation simply states that the gradient of E must be a linear combination of the gradients of the constraints. Combining the two conditions gives

$$\mathbf{J}\mathbf{J}^T \lambda = \dot{\mathbf{p}}_0 - \mathbf{J}\dot{\mathbf{q}}_0, \quad (7)$$

which is a matrix equation to be solved for λ . Then the camera parameter derivatives are given by

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}_0 + \mathbf{J}^T \lambda. \quad (8)$$

Finally, we must use the computed value of $\dot{\mathbf{q}}$ to update the camera state \mathbf{q} , a standard initial value problem. See Press *et al.* [20] for a discussion of the issues and a good assortment of numerical methods for ordinary differential equations. The very simplest method, *Euler's method*, employs the update formula

$$\mathbf{q}(t + \Delta t) = \mathbf{q}(t) + \Delta t \dot{\mathbf{q}}(t).$$

Although easy to implement, Euler's method is notoriously unstable and inaccurate. Use it at your own risk! In the interactive loop of through-the-lens control, drawing and input are interleaved with solver steps.

3.3 General quadratic objective functions

The restricted form of the objective function given in equation 6 is often adequate, but can cause problems: when the controls do not fully determine the camera's state, the task of accounting for the remaining degrees of freedom falls to the objective function. For example, if the camera is able to respond to the motion of a controlled point by a combination of tracking and panning, the objective function determines how much of each will take place. Because the error norm of equation 6 is the Euclidean distance in the camera's parameter space, rather than being intrinsic to the world-space camera motion, the behavior depends in a somewhat haphazard way on the choice of camera parameterization, and could even depend, for example, on the choice of linear and angular units of measure!

To allow such behavior to be controlled in a more rational way we make a reasonably straightforward generalization, allowing E to be any quadratic function of $\dot{\mathbf{q}}$, having

the form

$$E = \frac{1}{2} \dot{\mathbf{q}} \mathbf{M} \dot{\mathbf{q}} + \mathbf{b} \cdot \dot{\mathbf{q}} + c,$$

where \mathbf{M} is a matrix, typically symmetric and positive-definite, \mathbf{b} is a vector, and c is a scalar, none of them depending on $\dot{\mathbf{q}}$. Since E is quadratic, the problem remains linear, although the matrix equation to be solved becomes a bit more complex. The gradient of E becomes

$$\frac{\partial E}{\partial \dot{\mathbf{q}}} = \mathbf{M} \dot{\mathbf{q}} + \mathbf{b}.$$

Denoting the inverse of \mathbf{M} by $\mathbf{W} = \mathbf{M}^{-1}$, equation 7 assumes the form

$$\dot{\mathbf{p}}_0 = \mathbf{J} \mathbf{W} \mathbf{J}^T \lambda - \mathbf{J} \mathbf{W} \mathbf{b}. \quad (9)$$

It is also possible to solve for λ without obtaining the explicit inverse for \mathbf{M} by forming a larger linear system (see [8].)

Under this general linear/quadratic formulation, the camera's response to controls can be decoupled from the parameterization, for instance by letting \mathbf{M} be a *mass matrix* for the camera [13, 29, 31].

3.4 Multiple Points and Other Functions

Controlling more than one point involves a simple extension to the foregoing derivation. The matrix \mathbf{J} depends on \mathbf{x} , so each point being controlled yields a distinct version of equation 5. We combine the m equations into a single one by concatenating the derivative matrices to form a $2m \times n$ matrix, and concatenating the image velocities to form a $2m$ -long vector. From that point on, the derivation proceeds as above, to the solution of equation 7 for λ , which is now also a vector of length $2m$.

In addition to controlling image points directly, we would like to control functions of one or more points, such as image distance or orientation. In fact, to mix image-space and world-space controls we may want to control other functions of \mathbf{q} that do not involve the image at all, such as object-to-camera distance. Conceptually, this is not a difficult generalization to make: in equation 5, we simply interpret \mathbf{p} not as a literal point, but as the vector of quantities we wish to control. Matrix \mathbf{J} must then give the derivative of each controlled quantity with respect to each camera parameter. In practice, performing the derivative evaluations, indexing and other bookkeeping, etc., can become quite complex. See [14, 29] for general-purpose schemes that facilitate the handling of this kind of matrix-assembly problem. Although our own implementations are based on such a scheme, the camera control problem is sufficiently restricted in scope that this certainly is not necessary.

Many through-the-lens controls, such as point-to-point distance, can be expressed as functions of several image points' positions. The labor involved in implementing such controls can be greatly reduced through the use of the chain rule. For instance, consider a scalar

function of two image points $f(\mathbf{p}_1, \mathbf{p}_2)$. The derivative of f with respect to $\dot{\mathbf{q}}$ is

$$\frac{df}{d\dot{\mathbf{q}}} = \frac{\partial f}{\partial \mathbf{p}_1} \mathbf{J}_1 + \frac{\partial f}{\partial \mathbf{p}_2} \mathbf{J}_2,$$

where \mathbf{J}_1 and \mathbf{J}_2 are the derivative matrices for \mathbf{p}_1 and \mathbf{p}_2 , computed according to equation 4. The code that evaluates \mathbf{J} for image points need only be implemented once. Thereafter, just derivatives with respect to image points need be treated anew for each control. These tend to be simple, and as an added advantage, they are independent of the choice of the underlying camera parameterization.

3.5 Constrained Dragging and Soft Controls

When controls are added dynamically by the user, it is entirely possible for inconsistencies to arise, either because the degrees of control exceed the camera's degrees of freedom, or because some controls are in conflict, e.g. trying to move one point in two directions. These problems can be handled gracefully by employing a least-squares method to solve the matrix equation—see for example the conjugate gradient solver described in [20]—so that the error due to the inconsistency is distributed uniformly over the controls, in a least-squares sense.

Although the least-squares solution avoids disaster when conflicts arise, we have found that it is very helpful to permit the user to drag points and other features *subject to* the constraints imposed by existing controls, so that conflicts can never arise. We achieve this behavior by incorporating the dragged point's desired behavior into the objective function, rather than using a "hard" constraint to control it. The constrained optimization solution then resolves any conflicts strictly in favor of the hard constraints. Thus, for example, a point whose range of motion is restricted by the controls will move freely up to the limit of its travel, but no further. A simple way to implement such "soft" controls is to specify the desired camera motion $\dot{\mathbf{q}}_0$ according to the formula

$$\dot{\mathbf{q}}_0 = k_c \mathbf{J}^T (\mathbf{p}_c - \mathbf{p}), \quad (10)$$

where k_c is a constant and \mathbf{p}_c is the position of the cursor in image coordinates. Using this value to drive the system is similar to attaching a rubber band between \mathbf{p}_c and \mathbf{p} , inducing camera motion that causes \mathbf{p} to "chase" \mathbf{p}_c . Inserting this value of $\dot{\mathbf{q}}_0$ into equation 7 minimizes the mean squared difference between $\dot{\mathbf{q}}$ and $\dot{\mathbf{q}}_0$, subject to the constraints. Soft controls can be implemented more accurately, at the expense of greater complexity, by minimizing the squared difference between $\dot{\mathbf{p}}$ and a desired value $\dot{\mathbf{p}}_0$, subject to the constraints. To express this objective function, the general form given in equation 9 must be used.

A greatly simplified though much less powerful version of through-the-lens control is obtained by using soft controls only. Then, the constrained optimization of equation

6 collapses into an unconstrained optimization. For example, an image point could be dragged by using equation 10 directly to determine $\dot{\mathbf{q}}$.

3.6 Position Feedback

So far, we have cast the problem in terms of velocity control. The velocity signals that drive the control process may come from several sources. For example, during interactive dragging of a controlled image point, the velocity may represent an estimate of mouse velocity. In keyframing, the velocity represents the derivative of a known trajectory curve $\mathbf{p}_0(t)$. In both cases, position as well as velocity information is available. This extra information can be used to greatly improve tracking accuracy by preventing error accumulation and drift as velocity is integrated over time. We do this by the addition of a simple linear feedback term to our initial statement of the control requirement:

$$\dot{\mathbf{p}} = \dot{\mathbf{p}}_0 - k_f (\mathbf{p} - \mathbf{p}_0),$$

where k_f is a feedback constant, and \mathbf{p}_0 is the desired position for \mathbf{p} at the current time. When \mathbf{p} is on target, the feedback term vanishes, but if positional error exists, the velocity is biased in a direction that reduces the error. The feedback term carries straight through the derivation, leading to the following modified form for equation 7:

$$\mathbf{J} \mathbf{J}^T \lambda = \dot{\mathbf{p}}_0 + k_f (\mathbf{p}_0 - \mathbf{p}) - \mathbf{J} \dot{\mathbf{q}}_0. \quad (11)$$

3.7 Tracking a moving point

Until now, we have assumed that the world-space point \mathbf{x} is stationary. A small generalization makes it possible to accurately track a moving point. In keyframe animation, for example, this would allow moving points on objects to be tracked automatically. To make the generalization, we assume that the world-space point moves according to a known function $\mathbf{x}(t)$. In practice, we need only know the point's current position \mathbf{x} and velocity $\dot{\mathbf{x}}$. Since \mathbf{x} now depends on time, an additional term appears in equation 2, the chain-rule expression for $\dot{\mathbf{p}}$, accounting for the part of \mathbf{x} 's image velocity due to the motion of \mathbf{x} itself:

$$\dot{\mathbf{p}} = \mathbf{h}'(\mathbf{V}\mathbf{x}) \frac{\partial(\mathbf{V}\mathbf{x})}{\partial \mathbf{q}} \dot{\mathbf{q}} + \mathbf{h}'(\mathbf{V}\mathbf{x}) \mathbf{V} \dot{\mathbf{x}} \quad (12)$$

As before, the extra term carries through, adding an additional correction factor to the right hand side of equation 7, yielding

$$\mathbf{J} \mathbf{J}^T \lambda = \dot{\mathbf{p}}_0 + k_f (\mathbf{p}_0 - \mathbf{p}) - \mathbf{h}'(\mathbf{V}\mathbf{x}) \mathbf{V} \dot{\mathbf{x}} - \mathbf{J} \dot{\mathbf{q}}_0. \quad (13)$$

This formulation makes it possible to control the *image-space* motion of a point independently of its *world-space* motion. If the image point is pinned, the camera will move as necessary to maintain its position. Both the image point and the world point can be keyframed independently: the camera will move as required to achieve the desired image motion, regardless of the world-space motion of the point.

4 A Quaternion Camera

Having developed the through-the-lens equations in generic form, it remains to fill in the blanks. In the equations of the last section, the camera transformation was described in terms of an anonymous matrix \mathbf{V} depending on an anonymous parameter vector \mathbf{q} . To proceed, we must say what the function $\mathbf{V}(\mathbf{q})$ actually is. Then we must formulate the equations that are required to evaluate the image point derivative matrix \mathbf{J} . If we limit ourselves to image-space controls that can be expressed purely as functions of point positions, then the matrices \mathbf{V} and \mathbf{J} tell us everything we need to know about the camera.

As we noted in section 1, through-the-lens control hides the underlying camera parameterization from the user, so that most of the criteria by which a conventional camera model would be judged do not apply. The model we present in this section is unusual in that a quaternion is used to represent the camera's orientation; we choose it because of the quaternion's ability to represent arbitrary rotations free of singularities and other artifacts. The equations of section 3 are compatible with any camera model. If you prefer another one, the derivation in this section can still serve as a template for the general procedure.

4.1 The View Matrix

Our model employs a translation to specify the Lookfrom point and a quaternion to specify orientation. The view matrix \mathbf{V} called for by equation 1 is given by the matrix product

$$\mathbf{V} = \mathbf{P}(f)\mathbf{T}(t_x, t_y, t_z)\mathbf{Q}(q_w, q_x, q_y, q_z), \quad (14)$$

where \mathbf{P} is a matrix for perspective projection with focal length f , \mathbf{T} is the matrix for translation by $[t_x, t_y, t_z]$, and \mathbf{Q} is a *quaternion rotation matrix*, performing the rotation specified by the quaternion \mathbf{q} , with scalar part q_w and vector part $[q_x, q_y, q_z]$. The camera parameter vector \mathbf{q} is the length-8 vector formed by concatenating the transformation parameters, $[f, t_x, t_y, t_z, q_w, q_x, q_y, q_z]$.

The perspective matrix is a simple one, placing the focal point at the origin and the image plane at distance f from the origin along the z -axis, lying parallel to the xy -plane:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix}.$$

The translation matrix is the standard one:

$$\mathbf{T}(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The quaternion rotation matrix is a bit more complex. The

form given in [22],

$$\mathbf{Q} = 2 \begin{bmatrix} \frac{1}{2} - q_y^2 - q_z^2 & q_x q_y + q_w q_z & q_x q_z - q_w q_y & 0 \\ q_x q_y - q_w q_z & \frac{1}{2} - q_x^2 - q_z^2 & q_w q_x + q_y q_z & 0 \\ q_w q_y + q_x q_z & q_y q_z - q_w q_x & \frac{1}{2} - q_x^2 - q_y^2 & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}, \quad (15)$$

assumes that the quaternion has unit magnitude, i.e. that

$$|\mathbf{q}| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1.$$

Otherwise, \mathbf{Q} is not a pure rotation, and shapes will be distorted. This constraint on $|\mathbf{q}|$ means that the camera has only seven true degrees of freedom. To enforce the constraint, it is not sufficient simply to normalize \mathbf{Q} between iterations: in that case, the derivative matrix wouldn't "know" about the constraint, and the control solution would be incorrect. While it would be possible to add the constraint, in differential form, to the control solution, there is a much simpler alternative: in place of equation 15, we express \mathbf{Q} in a form that incorporates the normalization, so that quaternions \mathbf{q} and $\alpha\mathbf{q}$ specify the same transformation, for any scalar α . Under this scheme, we must still normalize \mathbf{q} from time to time to prevent the accumulation of numerical errors. The modified version of \mathbf{Q} is most simply expressed as the product

$$\mathbf{Q}_n = \frac{1}{|\mathbf{q}|^2} \hat{\mathbf{Q}},$$

where

$$\hat{\mathbf{Q}} = 2 \begin{bmatrix} \frac{|\mathbf{q}|^2}{2} - q_y^2 - q_z^2 & q_x q_y + q_w q_z & q_x q_z - q_w q_y & 0 \\ q_x q_y - q_w q_z & \frac{|\mathbf{q}|^2}{2} - q_x^2 - q_z^2 & q_w q_x + q_y q_z & 0 \\ q_w q_y + q_x q_z & q_y q_z - q_w q_x & \frac{|\mathbf{q}|^2}{2} - q_x^2 - q_y^2 & 0 \\ 0 & 0 & 0 & \frac{|\mathbf{q}|^2}{2} \end{bmatrix}$$

4.2 Evaluating \mathbf{J}

Employing the notation of section 3, the image coordinates corresponding to world point \mathbf{x} are given by

$$\mathbf{p} = \mathbf{h}(\mathbf{V}\mathbf{x}) = \mathbf{h}(\mathbf{P}\mathbf{Q}_n\mathbf{T}\mathbf{x}).$$

The rows of \mathbf{J} are formed by differentiating this expression with respect to each camera parameter in turn. To perform the differentiations, we note that each camera parameter influences exactly one matrix in the chain. Therefore, using the rule for differentiation of a product, the derivative of the chain with respect to a parameter is another chain, obtained by replacing the appropriate matrix by its element-by-element derivative. Thus, for example,

$$\frac{\partial \mathbf{V}\mathbf{x}}{\partial t_x} = \mathbf{P} \frac{\partial \mathbf{T}}{\partial t_x} \mathbf{Q}_n \mathbf{x},$$

and we obtain the row of \mathbf{J} corresponding to t_x from

$$\frac{\partial \mathbf{p}}{\partial t_x} = \mathbf{h}'(\mathbf{V}\mathbf{x}) \frac{\partial \mathbf{V}}{\partial t_x},$$

where $\mathbf{h}'(\mathbf{V}\mathbf{x})$ is as defined in equation 2, and where

$$\frac{\partial \mathbf{T}}{\partial t_x} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Differentiating each matrix with respect to each parameter on which it depends yields eight matrices in all. The matrix for $\partial \mathbf{T} / \partial t_x$ is given above—the other two derivatives of \mathbf{T} are likewise trivial. The derivative of \mathbf{P} with respect to its only parameter, f , is

$$\frac{\partial \mathbf{P}}{\partial f} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/f^2 & 0 \end{bmatrix}.$$

The four derivatives of \mathbf{Q}_n may be expressed compactly as

$$\frac{\partial \mathbf{Q}_n}{\partial q_w} = \frac{-2q_w}{|\mathbf{q}|^4} \hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2} \begin{bmatrix} q_w & q_z & -q_y & 0 \\ -q_z & q_w & q_x & 0 \\ q_y & -q_x & q_w & 0 \\ 0 & 0 & 0 & q_w \end{bmatrix},$$

$$\frac{\partial \mathbf{Q}_n}{\partial q_x} = \frac{-2q_x}{|\mathbf{q}|^4} \hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2} \begin{bmatrix} q_x & q_y & q_z & 0 \\ q_y & -q_x & q_w & 0 \\ q_z & q_w & -q_x & 0 \\ 0 & 0 & 0 & q_x \end{bmatrix},$$

$$\frac{\partial \mathbf{Q}_n}{\partial q_y} = \frac{-2q_y}{|\mathbf{q}|^4} \hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2} \begin{bmatrix} -q_y & q_x & -q_w & 0 \\ q_x & q_y & q_z & 0 \\ q_w & q_z & -q_y & 0 \\ 0 & 0 & 0 & q_y \end{bmatrix},$$

and

$$\frac{\partial \mathbf{Q}_n}{\partial q_z} = \frac{-2q_z}{|\mathbf{q}|^4} \hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2} \begin{bmatrix} -q_z & q_w & q_x & 0 \\ -q_w & -q_z & q_y & 0 \\ q_x & q_y & q_z & 0 \\ 0 & 0 & 0 & q_z \end{bmatrix}.$$

To evaluate \mathbf{J} , we need only implement functions that calculate each of these eight derivative matrices, along with the function that calculates $\mathbf{h}'(\mathbf{x})$. Standard matrix/vector operations are then used to produce the eight rows of \mathbf{J} .

5 Implementation and Examples

We have implemented through-the-lens control as part of a multi-view, direct manipulation testbed. The program is written in C++ on a Silicon Graphics Iris workstation and uses a toolkit which permits rapid evaluation of dynamically composed functions and their derivatives[14]. All of the examples in this paper can be specified interactively and run at interactive rates on a Silicon Graphics IRIS 4D/210 GTX.

We have experimented with a wide variety of through-the-lens controls including

- the position of a point on the screen,
- the distance between two points on the screen,
- the orientation of two points in the image,
- the ratio between two screen space distances.

All can be interactively specified and connected to vertices in the scene, can be made into hard or soft controls, can serve as constraints, and can be keyframed. Controls that do not have an obvious geometric method for direct manipulation, such as the last three on the list, can be connected to sliders.

The architecture of our system makes it easy to define new types of controls, although this must be done at compile time. Unlike finding new transforms, which entails solving systems of non-linear equations, defining new controls is easy to automate in a general and guaranteed manner since the only required mathematical manipulation is differentiation. We have built automatic code generation tools that facilitate defining new types of controls.

By adding the ability to place boundaries on the values of a control, we have been able to create several interesting through-the-lens features in our system, such as

- bounding a point within a region of the image,
- ensuring that an object does not become larger or smaller than a certain size,
- preventing an object from becoming too much bigger or smaller than another.

We use an active set technique[11] to extend the methods of section 3 to provide the capability of inequality constraints.

These through-the-lens controls work in concert with a variety of world-space controls. Because a camera is a first class object in our system, these controls can be applied to them as well as other objects in the scene. Multiple windows with cameras dynamically assigned to them make it easy to use world and image space controls together in composing an image.

Building on top of a general purpose facility for composing derivatives permits our implementation to exercise the full generality of the methods in section 3 by allowing us to solve simultaneously for camera and object parameters. Through-the-lens controls can therefore affect other objects in addition to the camera. Although removing the restriction that \mathbf{x} does not depend on \mathbf{q} does not require any change to the techniques presented, the pragmatic issues that arise in including parameters of objects other than the camera in \mathbf{q} are beyond the scope of this paper. These issues are discussed in [14, 13, 29]. They permit a unified approach to controlling and constraining all objects, including cameras.

When the state vector includes objects besides the camera, through-the-lens controls provide a way to couple the camera and scene objects. If a point on an object is pinned to a particular place in the image, as the object moves the camera will also change to maintain the constraint. Changing the camera will similarly alter the object. If the camera is locked in place, the object is restricted to locations where its image satisfies the through-the-lens requirements. Adjusting a through-the-lens control can cause both the camera and the scene objects to change,

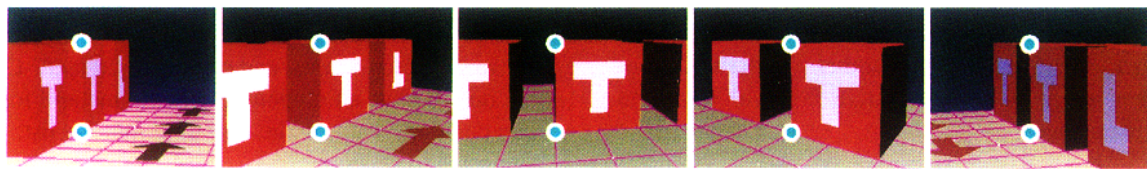


Figure 1: Multiple through-the-lens constraints: Multiple through-the-Lens point controls fixate two corners of the center cube. As the camera is translated along the faces of the cube (following the arrows on the floor), it rotates and zooms to maintain the constraints.

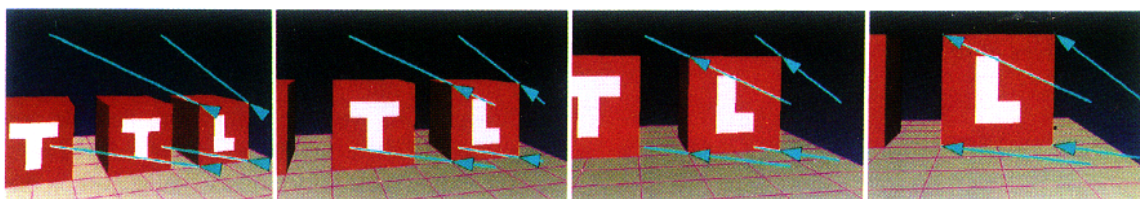


Figure 2: Through-the-lens keyframing: Through-the-lens controls are moved along keyframe paths. Each arrow grabs a corner of the cube and pulls it along a path in the image.

such controls permit manipulation of an object in terms of how it appears in the image. In a highly constrained environment, this can make it easy to achieve a desired effect when it is unclear how to do it by controlling the camera and other objects independently.

As a simple example of what through-the-lens controls can do, consider the role of the standard LOOKAT/-LOOKFROM/VUP camera model in our system. The ability to place points in the image and specify the orientations of line segments subsumes the need for this camera model. Although L/L/V is one of several camera models we have coded into our system², we typically prefer to use representations like the quaternion-based one in section 4 for their well-behavedness, using through-the-lens controls to point the camera. Even if the L/L/V representation is employed, the user is not restricted to specifying the view using these parameters.

The spacecraft example from the introduction exemplifies the use of through the lens controls to compose an image (Figure 3). Continuing with the example, the constraints used to position the spacecraft and planet can be maintained as the spacecraft flies past the planet to create a fly-by animation, either by coupling the state variables or using the tracking techniques of section 3.7. If the geometry of the scene isn't predetermined, through-the-lens control can help specify it. For example, consider creating a picture of the spacecraft flying by the planet and its moons. If we free the position of the craft, through-the-lens controls can move it so that its position in the image is maintained as we move the camera to find a view which shows the planets and the moons in a desirable manner.

Another use of through-the-lens controls is registering 3D models with photographs. This can be done by displaying a real image as a backdrop and pinning points on the synthesized image to their corresponding locations. Using

a least squares technique for overdetermined matrices can allow several points to be specified: the system will move towards the best fit (Figure 4). A viewing transform can be derived for registering 3 points [18], but through-the-lens techniques provide a general method for performing these manipulations.

6 Conclusion

As we gain more experience using through-the-lens control, we find more interesting controls and constraints to aid in the process of composing pictures and manipulating scenes. Other additions to through-the-lens manipulation might include using optimization and constraints to help compose images, developing an interface that makes it easier to specify both through-the-lens and world-space controls, inferring constraints to make manipulation easier, and providing a method of detecting and preventing unwanted occlusions. We are beginning to explore using through-the-lens techniques to connect 3D models to real photographs and live video. We are also considering how to use through-the-lens techniques to address issues in planning good camera motions for animations.

Through-the-lens techniques provide a method for manipulating the virtual camera by controlling and constraining image attributes. Interactive control techniques permit the user to control the virtual camera by directly manipulating the image as seen through the lens. The control techniques make it easy to enforce constraints on attributes of the image and scene. The techniques make it simple to implement a wide variety of constraints and controls.

Acknowledgements

This research was funded in part by Apple Computer, a fellowship from the Schlumberger Foundation, and an equipment grant from Silicon Graphics. We would like to thank Pete Wyckoff for the fractal planets, James Rehg

²Finding the derivatives of this matrix is not for the faint of heart — don't try it without a symbolic mathematics program.

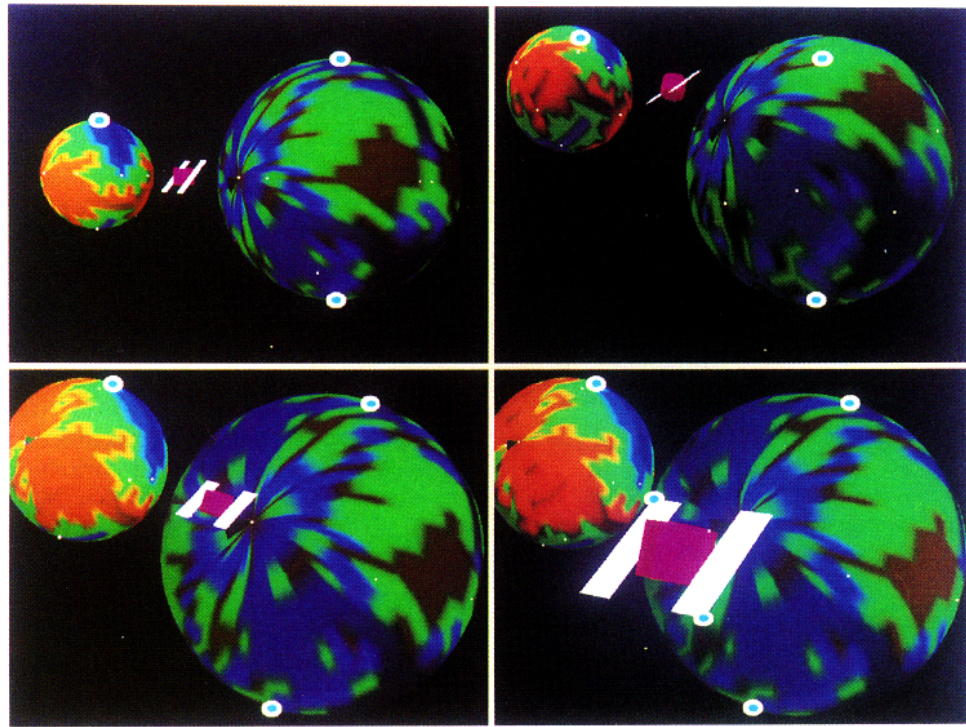


Figure 3: Composing an image with Through-the-Lens controls. Through-the-lens point controls are placed at the poles of the planet and the moon (upper left). The leftmost control is moved (upper right). The controls are adjusted further (lower left). After positioning the camera, through the lens controls are used to position the spacecraft (lower right).



Figure 4: Decorating the lab with Through-the-Lens controls: The rectangular polygon has the same dimensions as the tabletop. By interactively positioning the vertices to correspond with their respective corners, the virtual camera models the real camera.

and Michael Kass for providing some references, Steven Drucker and Tinsley Gaylean for helping renew our interest in camera placement, and Will Welch for providing caffeine and conversation.

References

- [1] Norman Badler, Kamran Manoocherhri, and David Baraff. Multi-dimensional input techniques and articulated figure positioning by multiple constraints. In *Proceedings of the 1986 Workshop on Interactive 3d Graphics*, pages 151–170, October 1986.
- [2] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22:179–188, 1988. Proceedings SIGGRAPH '88.
- [3] Jim Blinn. Where am I? What am I looking at? *IEEE Computer Graphics and Applications*, pages 76–81, July 1988.
- [4] Frederick Brooks. Walkthrough – a dynamic graphics environment for simulating virtual buildings. In *Proceedings of the 1986 Workshop on Interactive 3d Graphics*, pages 9–22, October 1986.
- [5] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3d rotation using 2d input devices. *Computer Graphics*, 22(4):121–130, August 1988. Proceedings SIGGRAPH '88.
- [6] PHIGS+ Committee. Phigs+ functional description, revision 3.0. *Computer Graphics*, 22(3):125–215, 1988.
- [7] Steven Drucker, Tinsley Gaylean, and David Zeltzer. CINEMA: a system for procedural camera movements. In *Proceedings of the 1992 Symposium on Interactive Computer Graphics*, pages 67–70, 1992.
- [8] Roger Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, 1987.
- [9] Sundaram Ganapathy. Decomposition of transformation matrices for robot vision. In *International Conference on Robotics*, pages 130–139, March 1984.
- [10] Donald Gennery. Stereo-camera calibration. In *Proc. DARPA Image Understanding Workshop*, pages 101–107, 1979.
- [11] Phillip Gill, Walter Murray, and Margret Wright. *Practical Optimization*. Academic Press, New York, NY, 1981.
- [12] Michael Gleicher. Briar - a constraint-based drawing program. In *CHI '92 Formal Video Program*, 1992. SIGGRAPH video review, in press.
- [13] Michael Gleicher and Andrew Witkin. Differential manipulation. *Graphics Interface*, pages 61–67, June 1991.
- [14] Michael Gleicher and Andrew Witkin. Snap together mathematics. In Edwin Blake and Peter Weisskirchen, editors, *Advances in Object Oriented Graphics 1: Proceedings of the 1990 Eurographics Workshop on Object Oriented Graphics*. Springer Verlag, 1991. Also appears as CMU School of Computer Science Technical Report CMU-CS-90-164.
- [15] David Lowe. Solving for the parameters of object models from image descriptions. In *Proc. DARPA Image Understanding Workshop*, pages 121–127, 1980.
- [16] Jock Mackinlay, Stuart Card, and George Robertson. Rapid controlled movement through a virtual 3d workspace. *Computer Graphics*, 24(4):171–176, August 1990.
- [17] Chris McGlone. Automated image-map registration using active contour models and photogrammetric techniques. In *Proceedings of the SPIE, Volume 1070*, January 1989.
- [18] Francis H. Moffitt. *Photogrammetry*. International Textbook Company, 1959.
- [19] John Platt and Alan Barr. Constraint methods for flexible models. *Computer Graphics*, 22:279–288, 1988. Proceedings SIGGRAPH '88.
- [20] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1986.
- [21] K. Schwidersky. *An Outline of Photogrammetry*. Pitman Publishing Corporation, first english edition, 1959.
- [22] Ken Shoemake. Animating rotations with quaternion curves. *Computer Graphics*, 19(3):245–254, July 1985.
- [23] Mark Surles. Interactive modeling enhanced with constraints and physics – with applications in molecular modeling. In *Proceedings of the 1992 Symposium on Interactive Computer Graphics*, pages 175–182, March 1992.
- [24] Konstantinos Tarabamis, Roger Tsai, and Peter Allen. Automated sensor planning for robotic vision tasks. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 76–82, April 1991.
- [25] Russell Turner, Francis Balaguer, Enrico Gobbetti, and Daniel Thalmann. Physically-based interactive camera motion using 3d input devices. In N. M. Patrikalakis, editor, *Scientific Visualization of Physical Phenomena: Proceedings of CG International 1991*, pages 135–145, Tokyo, 1991. Springer-Verlag.
- [26] Colin Ware and Steven Osborne. Exploration of virtual camera control in virtual three dimensional environments. *Computer Graphics*, 24(2):175–184, March 1990. Proceedings 1990 Symposium on Interactive 3D Graphics.
- [27] William Welch, Michael Gleicher, and Andrew Witkin. Manipulating surfaces differentially. In *Proceedings, Compugraphics '91*, September 1991. Also appears as CMU School of Computer Science Technical Report CMU-CS-91-175.
- [28] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(4):225–232, July 1987.
- [29] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–21, March 1990. Proceedings 1990 Symposium on Interactive 3D Graphics.
- [30] Andrew Witkin, Michael Kass, Demetri Terzopoulos, and Kurt Fleischer. Physically based modeling for vision and graphics. In *Proc. DARPA Image Understanding Workshop*, pages 254–278, 1988.
- [31] Andrew Witkin and William Welch. Fast animation and control of non-rigid structures. *Computer Graphics*, 24(4):243–252, August 1990. Proceedings SIGGRAPH '90.

Chapter 5

Acknowledgements

This work has been funded (in part) by the European Commission under grant agreement IRIS (FP7-ICT-231824).