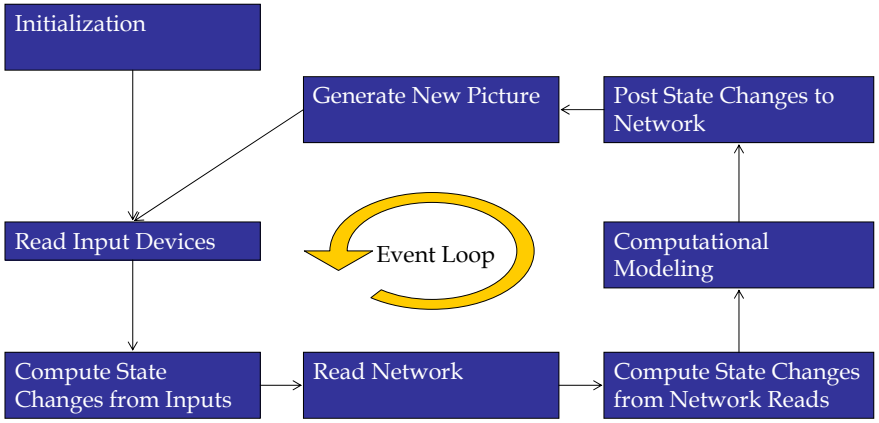
System Design

448430
Spring 2009
4/13/2009
Kyoung Shin Park
Multimedia Engineering
Dankook University

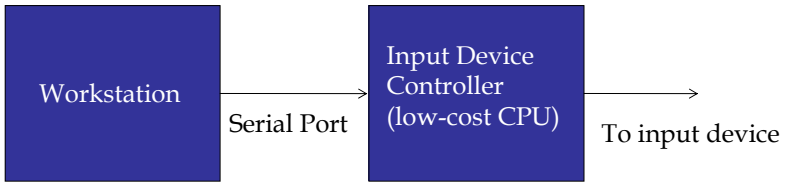
What does NVE software look like?

- Thread
  - One Thread, Multiple Threads
- Important Subsystems
  - Real-Time Rendering
    - Polygon Culling
    - Level of Detail Processing
  - Real-Time Collision Detection and Response
  - Computational Resource Management
  - Interaction Management

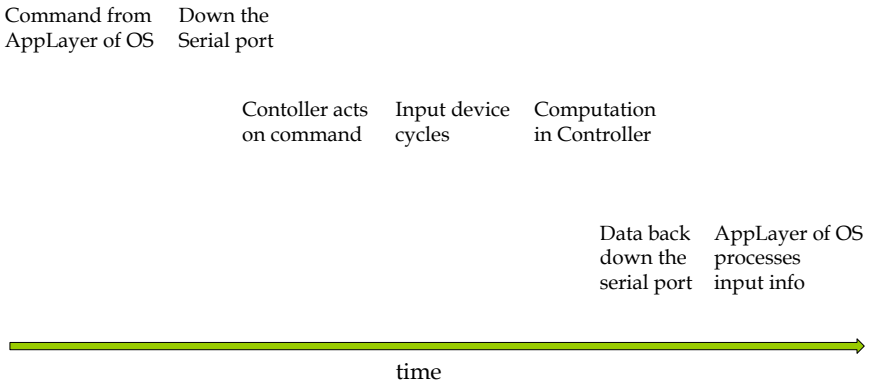
Single Thread



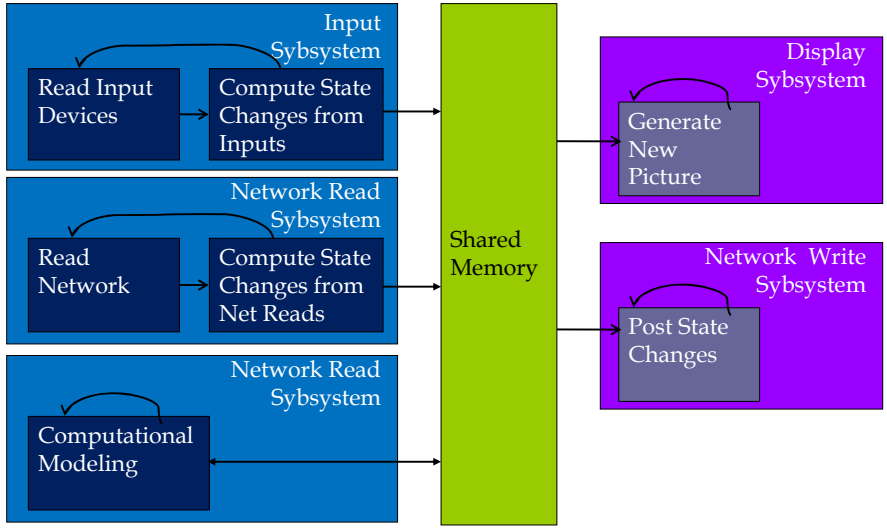
Input Device Connections to the Workstation



Sources for Lag - Input Devices



Multiple Subsystems, Multiple Threads

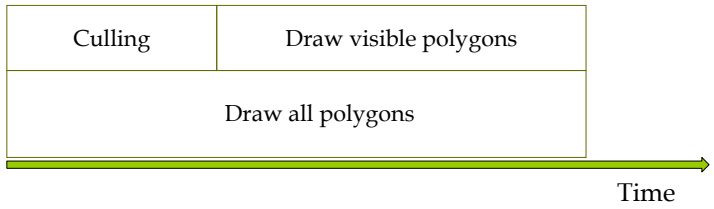


Real-Time Rendering

- Key problem
  - Limitations in the performance of graphics hardware
- Polygon culling
  - We try to use available CPU cycles to throw away most of our 3D model before we send it through the graphics pipeline …
- Level of detail processing

Polygon Culling

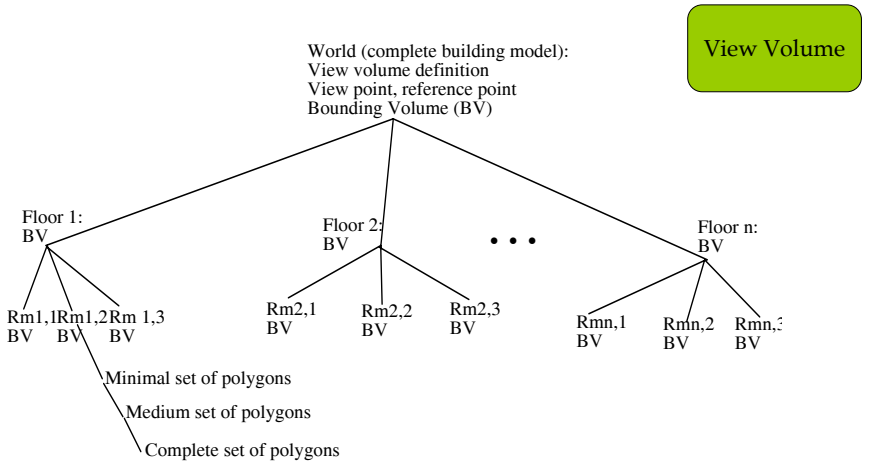
- Reduce the number of processed polygons
  - Determine which polygons do not need to be drawn
- A wealth of research and methods exists
- Assumes that the underlying 3D model remains quite static
  - Changes in the model => changes in the culling data structures



Hierarchical Data Structures for Polygon Culling

- The classic reference for this is the 1976 paper by James Clark "Hierarchical Geometric Models for Visible Surface Algorithms".
- The idea in the Clark paper is to build a hierarchical data structure for the displayable world ...

Hierarchical Data Structure for the Displayable World



Part 1 of the Clark Paper

- We build a data structure that allows us to rapidly throw away most of the data for the world.
- We test the hierarchical bounding volumes to see if they are contained or partially contained in the current orientation of the view volume.
- If they are and are leaves in the tree, we draw them or continue our traversal.

Part 2 of the Clark Paper

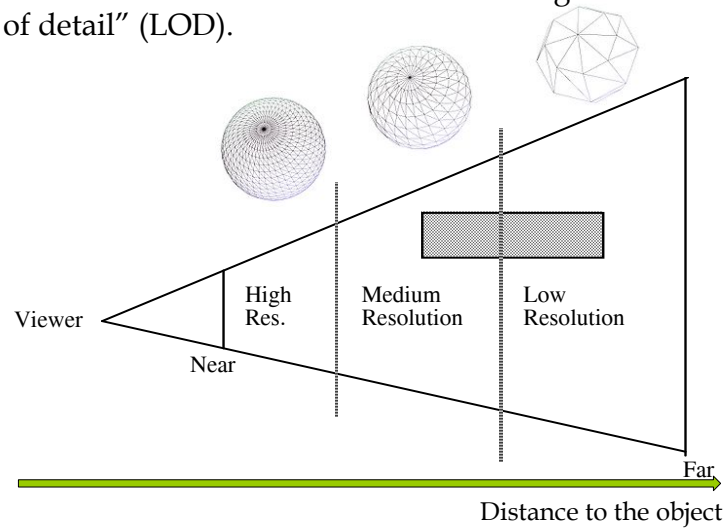
- We only send the minimal required description of our object through the graphics pipeline.
- We use the distance from the viewer to the BV/object to determine which resolution of our objects to display, basically the pixel coverage of the final drawn object.
- This presupposes multiple resolution versions of each room.

View Volume

- We can decide rather quickly whether our BV is in the High, Medium or Low resolution sections.
  - Transform the BVs with the far clipping plane moved closer.
  - Clip into memory and make the decision (or perform this test in the CPU, if there are spare cycles).
  - We will end up with a list of BVs that should be displayed and a resolution at which they should be displayed.

Level Of Detail (LOD)

- This is the view volume cut into three regions or "levels of detail" (LOD).

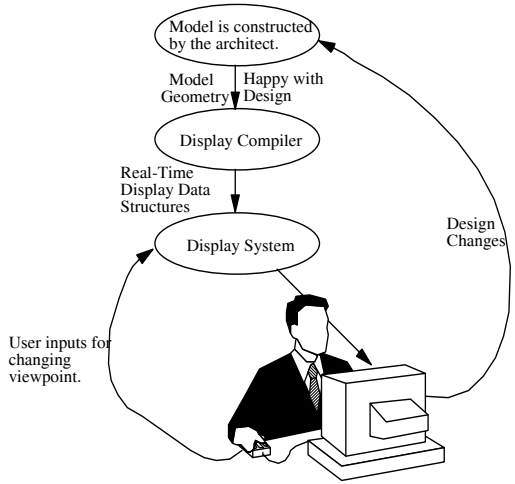


How do we compute different LODs for our models?

- This depends on the origins of our models...
- By hand - we do this by hand in our modeling tool, throwing away small polygons for the LOW resolution versions of our models.
  - Some modeling tools will do this semi-automatically. They give you a cut at it and you can add polygons back in.
- Triangular decimation - a triangular mesh is fitted to your model and an appropriate algorithm is used to reduce the total number of triangles in the model.
  - There is some very nice work on this by Greg Turk and Hughe Hoppe and others...

Airey, Rohlf & Brooks Paper

- Precomputation of a hierarchical data structure for a building.



Airey, Rohlf & Brooks Paper - Study of Architectural DBs

- The model is changed much less often than the viewpoint.
  - Means pre-processing the database (display compiler) is possible.
  - We could possibly make changes to the big data structure as we added new building components.
  - Perhaps the real-time data structure is updated in parallel. During the "think time" of the engineer at the workstation.

Airey, Rohlf & Brooks Paper - Study of Architectural DBs

- Many buildings have high average depth complexity.
  - Any image computed from an interior viewpoint will have many surfaces covering each pixel.
  - Much of the model contributes NOTHING to any given image.

Airey, Rohlf & Brooks Paper - Study of Architectural DBs

- Most polygons are axial.
  - They are parallel to 2 of the coordinate axes.
  - Most polygons are rectangles.

Airey, Rohlf & Brooks Paper - Study of Architectural DBs

- The set of polygons that appears in each view changes slowly as the viewpoint moves.
  - Except when crossing certain thresholds.
    - Doors, windows --> portals.

Airey, Rohlf & Brooks Paper - Study of Architectural DBs

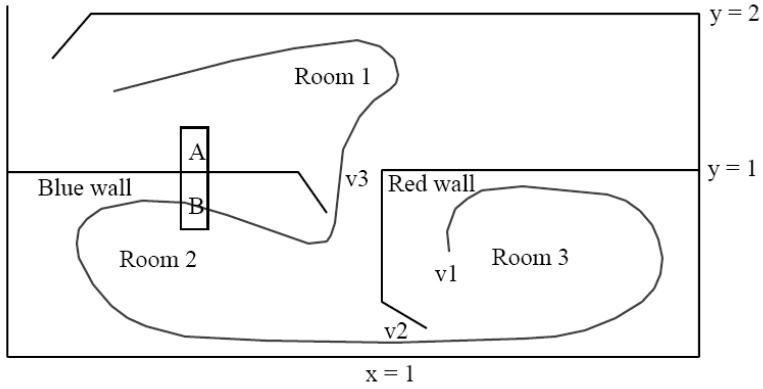
- We can possibly have the notion of a "working set" of bounding volumes.
  - Based on a viewpoint.
  - Also see this in the Clark paper.
  - We could just incrementally add/subtract branches of our tree based on view point changes.

Airey, Rohlf & Brooks Paper - Study of Architectural DBs

- This means that when we organize our data, our data inside one bounding volume should be co-located in CPU memory.
  - To make best use of the virtual memory system.

Portals and Viewpoints ...

- We have the notion of viewpoints at portals being indicators that we need to swap in major new blocks of data.



Airey, Rohlf & Brooks Paper - Study of Architectural DBs

- Large planar surfaces are often structured into multiple, co-planar levels for modeling purposes, shading purposes and realism purposes.
  - Large walls that cross several rooms might be stored as multiple polygons of different color.
  - Perhaps BV-wise, we could use the larger wall better than the smaller components.
  - Notion of somehow taking advantage of such dividing planes in building our tree structure.

Airey, Rohlf & Brooks Paper - Study of Architectural DBs

- For viewpoints inside the building, the role of surface inter-reflections in shading calculations is very important for spatial comprehension.
  - In the Airey system, there is an adaptive radiosity display algorithm.
  - When the viewpoint is not changing, the better radiosity view id displayed.
    - An adaptive system. Put up more detail when the system is not moving.

Model Space Subdivision

- How UNC builds its data structures for display.
- The use a display compiler.
  - Automatically subdivide their database into cells based on the union of "potentially visible sets" (PVSs) for any viewpoint in a cell.
  - Viewpoint --> which cell to display
  - That cell contains potentially visible information.

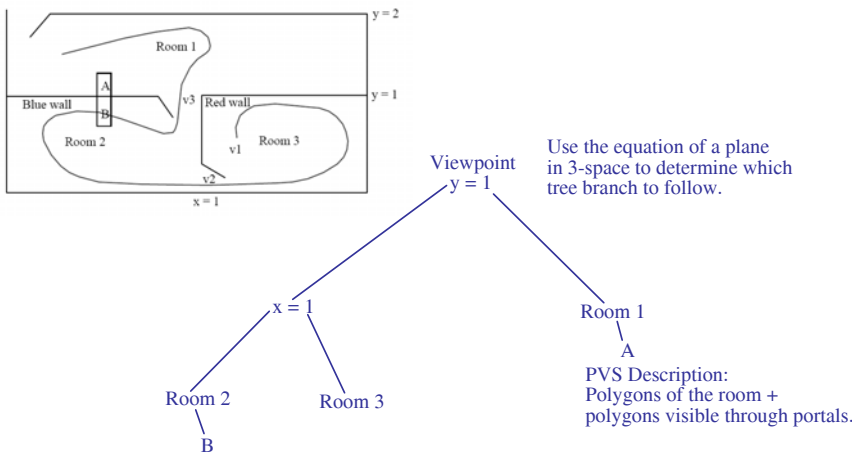
Model Space Subdivision

- A cell is a room plus any potentially visible polygons, polygons visible through portals.
- Computing PVSs is a hard problem.
- For any viewpoint, we must display the polygons for the room plus any possibly visible ones through any doors/portals.

Binary Space Partitioning

- Airey used a BSP-tree as the data structure for his building models.
- His paper describes how to automatically choose dividing planes.
- Use the biggest polygon, the ones with the most occlusion potential.

Binary Space Partitioning



Partition Priority for Polygons

- One of the key problems in model space subdivision is determining which are the best planes for splitting the geometric database.
- Airey came up with the idea of a "partition priority" for any polygon.

partition priority for any polygon (used to determine best plane for splitting the database.) = 0.5 * occlusion + 0.3 * balance + 0.2 * split

The biggest known polygons with best occlusion potential are weighted the most.

BSP-tree balance, i.e. 1/2 polygons are on each side of the dividing polygon.

Sometimes must cut polygons by the dividing plan (want to minimze that.)

Summary of Model Subdivision Results

§	Polygons§	Cells§	Polys/ Cell§	Polys/ Cell§	Speed-up§	Speed-up§
§	§	§	Average§	Max.§	Average§	Min.§
Sitt. Hall§	7125§	269§	230§	2195§	30.98§	3.25§
Lobby§	3949§	54§	466§	2550§	8.47§	1.55§
Church 1§	7812§	108§	291§	2055§	26.85§	3.8§
Church 2§	6037§	16§	1887§	3477§	3.2§	1.74§

Optimal Number of Polygons Per Cell?

- What is the optimal number of polygons per cell?
  - 200 to 300 polygons per cell seems good for this system.
- The number of polygons per cell is determined by the graphics hardware's fill capability and by the CPU capability.
  - CPU capability is how long it takes to throw things away.
  - i.e. how long does it take us to compute which cells to display.

Papers Useful for Walkthrough

- 1976 CACM Clark, James H. "Hierarchical Geometric Models for Visible Surface Algorithms"
- Fuchs "Near-Real-Time Shaded Display of Rigid Objects" - BSP-tree fundamentals. SIGGRAPH
- Brooks. 1986 Workshop on Interactive 3D Graphics - Early thoughts on walkthroughs. "Walkthrough - A Dynamic Graphics System for Simulating Virtual Buildings"
- Airey, Rohlf & Brooks. 1990 Symposium on Interactive 3D Graphics paper. "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments"

Real-Time Collision Detection & Response

- Movement through our NVEs requires that we have some way to determine if we have collided with the surfaces in our world so that we can stop our movement or react to the collision.
- Interactivity in our NVEs requires that we have some way of reaching out and touching an object in our VE, being able to determine what we touched and then being able to react.
- Both Movement & Interactivity require real-time collision detection & response.
  - And that is a fine way to consume processor cycles ...

Real-Time Collision Detection & Response

- The problem of collision detection has been explored in the literature of computer graphics, robotics, computational geometry, computer animation, and physically-based modeling.
- Numerous approaches have been proposed.
  - Geometric reasoning[DK90]
  - Bounding volume hierarchy [Hub96]
  - Spatial representation [GASF94, NAT90]
  - Numerical methods[Cam97, GJK88]
  - Analytical methods[LM95, Sea93]
- However, many of these algorithms do not satisfy the demanding requirements of general-purpose collision detection for networked virtual environments.

Real-Time Collision Detection & Response

- By unifying several techniques from previous work, exploiting temporal and spatial coherence, and utilizing locality and hierarchical data structures, the research team at UNC has developed several collision detection algorithms and systems targeted toward large-scale, interactive simulation environments, including I-COLLIDE, RAPID, and V-COLLIDE.
- Public domain code is available for ftp at www.cs.unc.edu/ geom.
- Another recent effort is undertaken by a Netherland research team in developing another collision detection system called SOLID. The code is available at www.win.tue.nl/cs/tt/gino/solid/.

Real-Time Collision Detection & Response

- [Cam97] S. Cameron. Enhancing gjk: Computing minimum and penetration distance between convex polyhedra. Proceedings of International Conference on Robotics and Automation, 1997.
- [CLMP95] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In Proc. of ACM Interactive 3D Graphics Conference, pages 189{196, 1995.
- [DK90] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of pre-processed polyhedra { A uni ed approach. In Proc. 17th Internat. Colloq. Automata Lang. Program., volume 443 of Lecture Notes Comput. Sci., pages 400{413. Springer-Verlag, 1990.
- [GASF94] A. Garcia-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. IEEE Comput. Graph. Appl., 14:36{43, May 1994.

Real-Time Collision Detection & Response

- [GJK88] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between objects in three-dimensional space. IEEE J. Robotics and Automation, vol RA-4:193{203, 1988.
- [GLM96] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In Proc. of ACM Siggraph'96, pages 171{180, 1996.
- [HLC+97] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-collide: Accelerated collision detection for vrml. In Proc. of VRML Conference, pages 119{125, 1997.
- [Hub96] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. ACM Trans. Graph., 15(3):179{210, July 1996.

Computational Resource Management

- In our multi-threaded NVE system, there is one fundamental question above all:
  - How do we make sure the threads of our NVE don't hog the processors?

Interaction Management

- What does our sensor to action pathway look like?
  - Issues with sensor to abstraction to resolution…