

ECE 595 / CS 491 / CS 591
**Real-Time Rendering &
Graphics Hardware**

Pradeep Sen
Advanced Graphics Lab

Class 4
January 29, 2007

Last time

- Finish the rendering pipeline
 - Rasterization
 - Texturing
 - Framebuffer operations
- Handed out the project

Today

- Q/A session
- Application-side optimizations and data structures

Questions?

- Office hours Tuesday 2 – 4pm in ECE 225C

Back to the application

- Last week, we covered the graphics pipeline
- Now we go back to the beginning...

Application-side optimizations

- We will focus on optimizations for rendering
- In short: don't render what cannot be seen
 - Don't render stuff that is off screen (view-frustum culling)
 - Don't render stuff that is hidden by something else (occlusion culling)

Don't we do this already?

- Don't render stuff that is off screen
 - Clipping algorithm
- Don't render stuff that is hidden by something else
 - z-buffer algorithm
- The problem: inefficiency!
- Both algorithms scale linearly with scene complexity!

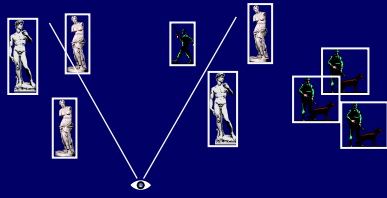
The algorithms

- Quickly eliminate geometry that cannot be seen
- We want to be conservative. If there is a chance the object can be seen, render it.
- We don't throw it out unless we are certain it does not appear on the screen
- This create what is known as a potentially visible set (PVS)
- Still use clipping and z-buffer for accurate rendering

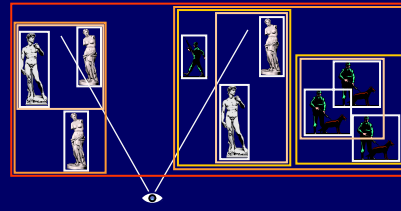
Bounding volume hierarchy

- Bound complex objects by a simple geometrical representation
- If bounding geometry is not visible, the object is not visible

Example



Example

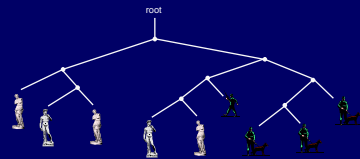


Hierarchical Bounding Box Algorithm

- Requires spatial sorting of scene objects
- If objects are not spatially coherent in the data structure, then bounding boxes will be very loose
- We can use the scene graph, which describes the scene

Hierarchical Bounding Box Algorithm

- Represent the bounding box hierarchy as a tree data structure



Hierarchical Bounding Box Algorithm

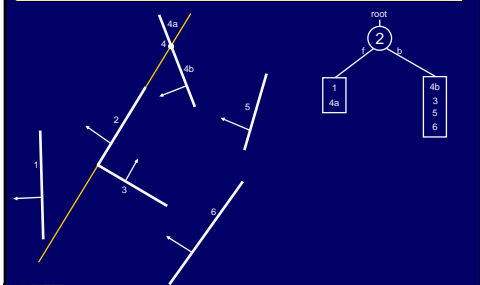
```
void HBoundingBoxRender(ptr* root, View view) {
    if (!testBbox(root->bbox, view))
        return;

    if (root->isPrimitive) {
        render(root->geometry);
    }
    else {
        // root is a box in the hierarchy
        for (i = 0; i < root->num_child; i++) {
            HBoundingBoxRender(root->child[i]);
        }
    }
}
```

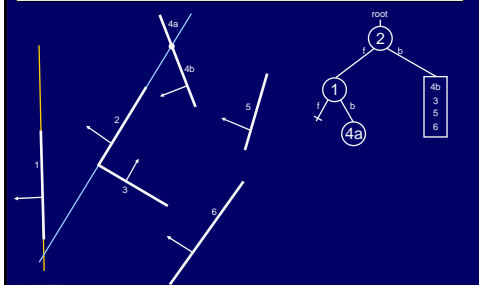
Binary space partitioning

- Another way to subdivide space

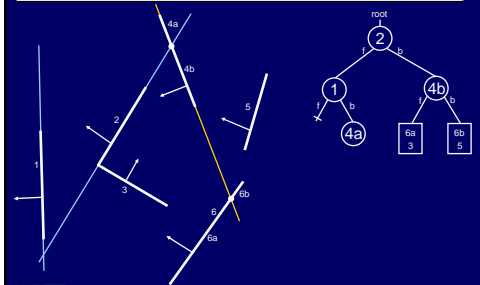
Binary space partitioning



Binary space partitioning



Binary space partitioning



Binary space partitioning

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

Binary space partitioning

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

Binary space partitioning

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

BSP construction

- Happens once for a static scene
- Must be updated every time the scene geometry changes
- However, it is possible to add new objects to the structure
- Now that we have created it, let us traverse it during rendering

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

BSP traversal algorithm (back-to-front)

- For each node of the tree:
 - visit far branch first
 - render current node
 - visit front branch

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

BSP traversal algorithm (back-to-front)

```

void BackToFrontBSP(ptr* root, View view) {
    if (!root)
        return; // return if NULL

    if (positiveSide(root, view) {
        backToFrontBSP(root->negBranch, view);
        render(root->geometry);
        backToFrontBSP(root->posBranch, view);
    }
    else {
        backToFrontBSP(root->posBranch, view);
        render(root->geometry);
        backToFrontBSP(root->negBranch, view);
    }
}
    
```

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

BSP traversal algorithm (back-to-front)

rendering order:
1 4a 2 6b 5 4b 6a 3

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

BSP traversal algorithm (back-to-front)

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

BSP traversal algorithm (back-to-front)

- No z-buffer needed!
- Surfaces are automatically ordered from back to front
- Essentially it is an implementation of the painter's algorithm

Real-time Rendering & Graphics Hardware
Pradeep Sen
Class 4 - January 29, 2007

Painter's algorithm

- As long as you paint the surfaces from back to front order, visible surface determination is automatic

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

Problems

- Pixels might be overdrawn over and over again
- Inefficient!
- In graphics we call this overdraw the "depth complexity" of the scene

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

Depth complexity

$d = \# \text{ frags rendered} / \# \text{ pixels in the image}$

- Essentially tells you how many times, on average, each pixel was drawn

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

BSP traversal algorithm (front-to-back)

- For each node of the tree:
 - visit front branch first
 - render current node
 - visit far branch
- To avoid clobbering a surface with something behind it, check to see if anything has been drawn at that pixel before writing

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

BSP traversal algorithm (front-to-back)

```

void frontToBackBSP(ptr* root, View view) {
    if (!root)
        return; // return if NULL

    if (positiveSide(root, view) {
        frontToBackBSP(root->posBranch, view);
        render(root->geometry);
        frontToBackBSP(root->negBranch, view);
    }
    else {
        frontToBackBSP(root->negBranch, view);
        render(root->geometry);
        frontToBackBSP(root->posBranch, view);
    }
}
  
```

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

BSP traversal algorithm (front-to-back)

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

BSP traversal algorithm (front-to-back)

- Before a pixel is set we have to test to see if there is something already there
- Depth complexity same as before
- With added work for checking, the front-to-back could be more expensive than back-to-front traversal?
- Solution: need a fast way to check which pixels have been already set

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

Dynamic screen

- Maintain an array of pointers the height of the screen
- Each pointer will indicate a linked list that represents continuous segments of unlit pixels
- Each element of the list has three fields: xMin, xMax, and pointer to next item
- When rasterizing polygons, each span is compared to the list at specific y position to see which pixels should be drawn

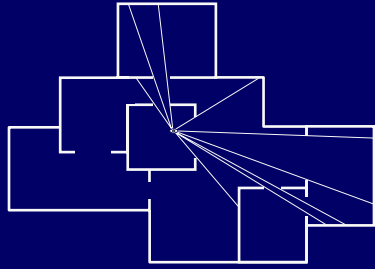
AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

Cells and portals

- Keep track of the doorways in rooms to precompute visibility

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 4 - January 29, 2007

Cells and portals



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 4 – January 29, 2007

Other approaches

- LOD: simplify the geometry of objects far away so that you do not have to render as many triangles
- Careful when you switch resolution to avoid “popping”



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 4 – January 29, 2007

Reading

- Article: “The Wizardry of Id,” *IEEE Spectrum*
- Article: “Front-to-Back Display of BSP Trees,” Gordon & Chen
- Moller & Haines, Chapter 9 through 9.5



Real-time Rendering & Graphics Hardware
Pradeep Sen

Class 4 – January 29, 2007