Lecture 26 of 41

Picking
Videos 5: More CGA

William H. Hsu
Department of Computing and Information Sciences, KSU

Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:

Next class: §8.3 – 8.4, 4.2, 5.0, 5.6, 9.1, Eberly 2e
Lecture Outline

- Reading for Last Class: Chapter 6, Esp. §6.1, Eberly 2e
- Reading for Today: Chapter 7, §8.4, Eberly 2e
- Reading for Next Class: §8.3 – 8.4, 4.2, 5.0, 5.6, 9.1, Eberly 2e
- Last Time: Adaptive Spatial Partitioning
  - Visible Surface Determination (VSD) revisited
  - Constructive Solid Geometry (CSG), Binary Space Partitioning (BSP)
  - Quadtrees (2-D) & octrees (3-D)
- Today: Picking
  - OpenGL modes: rendering (default), feedback, selection
  - Name stack
  - Hit records
  - Rendering in selection mode
  - Using selection buffer
  - Color coding to keep track of what has been picked, what to do
- Next Class: Interaction Handling
### Where We Are

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Lab 4a: Animation Basics</td>
<td>Flash animation handout</td>
</tr>
<tr>
<td>22</td>
<td>Animation 2: Rotations, Dynamics, Kinematics</td>
<td>Chapter 17, esp. §17.1 – 17.2</td>
</tr>
<tr>
<td>23</td>
<td>Demos 4: Modeling &amp; Simulation, Rotations</td>
<td>Chapter 10, 13, §17.3 – 17.5</td>
</tr>
<tr>
<td>24</td>
<td>Collisions 1: axes, OBBs, Lab 4b</td>
<td>§2.4.3, 8.1, GL handout</td>
</tr>
<tr>
<td>25</td>
<td>Spatial Sorting: Binary Space Partitioning</td>
<td>Chapter 6, esp. §6.1</td>
</tr>
<tr>
<td>26</td>
<td>Demos 5: More CGA: Picking; HW/Exam</td>
<td>Chapter 7, §8.4</td>
</tr>
<tr>
<td>27</td>
<td>Lab 5a: Interaction Handling</td>
<td>§8.3 – 8.4, 4.2, 5.0, 5.6, 9.1</td>
</tr>
<tr>
<td>28</td>
<td>Collisions 2: Dynamic, Particle Systems</td>
<td>§9.1, particle system handout</td>
</tr>
<tr>
<td></td>
<td>Exam 2 review; Hour Exam 2 (evening)</td>
<td>Chapters 5 – 6, 7 – 8, 12, 17</td>
</tr>
<tr>
<td>29</td>
<td>Lab 5b: Particle Systems</td>
<td>Particle system handout</td>
</tr>
<tr>
<td>30</td>
<td>Animation 3: Control &amp; IK</td>
<td>§5.3, CGA handout</td>
</tr>
<tr>
<td>31</td>
<td>Ray Tracing 1: intersections, ray trees</td>
<td>Chapter 14</td>
</tr>
<tr>
<td>32</td>
<td>Lab 6a: Ray Tracing Basics with POV-Ray</td>
<td>RT handout</td>
</tr>
<tr>
<td>33</td>
<td>Ray Tracing 2: advanced topic survey</td>
<td>Chapter 15, RT handout</td>
</tr>
<tr>
<td>34</td>
<td>Visualization 1: Data (Quantities &amp; Evidence)</td>
<td>Tutte handout (1)</td>
</tr>
<tr>
<td>35</td>
<td>Lab 6b: More Ray Tracing</td>
<td>RT handout</td>
</tr>
<tr>
<td>36</td>
<td>Visualization 2: Objects</td>
<td>Tutte handout (2 &amp; 4)</td>
</tr>
<tr>
<td>37</td>
<td>Color Basics, Term Project Prep</td>
<td>Color handout</td>
</tr>
<tr>
<td>38</td>
<td>Lab 7: Fractals &amp; Terrain Generation</td>
<td>Fractals/Terrain handout</td>
</tr>
<tr>
<td>39</td>
<td>Visualization 3: Processes; Final Review 1</td>
<td>Tutte handout (3)</td>
</tr>
<tr>
<td>40</td>
<td>Project presentations 1: Final Review 2</td>
<td>–</td>
</tr>
<tr>
<td>41</td>
<td>Project presentations 2</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Final Exam</td>
<td>Ch. 1 – 8, 10 – 15, 17, 20</td>
</tr>
</tbody>
</table>

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review; and the green-shaded entry, that of the term project.

Green, blue and red letters denote exam review, exam, and exam solution review dates.
Acknowledgements:
Collisions, Data Structures, Picking

Steve Rotenberg
Visiting Lecturer
Graphics Lab
University of California – San Diego
CEO/Chief Scientist, PixelActive
http://graphics.ucsd.edu

Glenn G. Chappell
Associate Professor
Department of Computer Science
University of Alaska Fairbanks
http://www.cs.uaf.edu/~chappell/

Edward Angel
Professor Emeritus of Computer Science
Founding Director, ARTS Lab
University of New Mexico
http://www.cs.unm.edu/~angel/
Review [1]:
Tree Representations for Scenes

- **Scene Graphs**
  - Organized by how scene is constructed
  - Nodes hold objects

- **Constructive Solid Geometry (CSG) Trees**
  - Organized by how scene is constructed
  - Leaves hold 3-D primitives
  - Internal nodes hold set operations

- **Binary Space Partitioning (BSP) Trees**
  - Organized by spatial relationships in scene
  - Nodes hold facets (in 3-D, polygons)

- **Quadtrees & Octrees**
  - Organized spatially
  - Nodes represent regions in space
  - Leaves hold objects
We think of scene graphs as looking like the tree on the left. However, it is often convenient to implement them as shown on the right.

- Implementation is a B-tree.
- Child pointers are first-logical-child and next-logical-sibling.
- Then traversing the logical tree is a simple pre-order traversal of the physical tree. This is how we draw.
Review [3]:

Binary Space Partitioning (BSP) Tree

- **BSP tree**: type of binary tree
  - Nodes can have 0, 1, or two children
  - Order of child nodes matters, and if a node has just 1 child, it matters whether this is its left or right child

- **Each node holds a facet**
  - This may be only part of a facet from original scene
  - When constructing a BSP tree, we may need to split facets

- **Organization**
  - Each facet lies in a unique plane
    - In 2-D, a unique line
  - For each facet, we choose one side of its plane to be “outside”
    - Other direction: “inside”
    - This can be the side the normal vector points toward
  - Rule: For each node
    - Its left descendant subtree holds only facets “inside” it
    - Its right descendant subtree holds only facets “outside” it

Adapted from slides © 2004 G. G. Chappell, UAF
Suppose we are given the following (2-D) facets and “outside” directions:

- We iterate through the facets in numerical order
  - Facet 1 becomes the root
  - Facet 2 is inside of 1
  - Thus, after facet 2, we have the following BSP tree:

Facet 3 is partially inside facet 1 and partially outside.
- We split facet 3 along the line containing facet 1
- The resulting facets are 3a and 3b
- They inherit their “outside” directions from facet 3

We place facets 3a and 3b separately
- Facet 3a is inside facet 1 and outside facet 2
- Facet 3b is outside facet 1

The final BSP tree looks like this:
Review [5]:
BSP Tree Traversal Example

- Procedure:
  - For each facet, determine on which side of it the observer lies.
  - Back-to-front ordering: Do an in-order traversal of the tree in which the subtree opposite from the observer comes before the subtree on the same side as the observer.

- Our observer is inside 1, outside 2, inside 3a, inside 3b.

- Resulting back-to-front ordering: 3b, 1, 2, 3a.
- Is this really back-to-front?
Order in which we iterate through the facets can matter a great deal

- Consider our simple example again
- If we change the ordering, we can obtain a simpler BSP tree

If a scene is not going to change, and the BSP tree will be used many times, then it may be worth a large amount of preprocessing time to find the best possible BSP tree.

Adapted from slides © 2004 G. G. Chappell, UAF
In general

- **Quadtree**: tree in which each node has at most 4 children
- **Octree**: tree in which each node has at most 8 children
- **Binary tree**: tree in which each node has at most 2 children

In practice, however, we use “quadtree” and “octree” to mean something more specific

- Each node of the tree corresponds to a square (quadtree) or cubical (octree) region
- If a node has children, think of its region being chopped into 4 (quadtree) or 8 (octree) equal subregions
- Child nodes correspond to these smaller subregions of parent’s region
- Subdivide as little or as much as is necessary
- Each internal node has exactly 4 (quadtree) or 8 (octree) children
Review [8]:
Quadtree Construction Example

- Root node of quadtree corresponds to square region in space
  - Generally, this encompasses entire “region of interest”

- If desired, subdivide along lines parallel to the coordinate axes, forming four smaller identically sized square regions
  - Child nodes correspond to these

- Some or all of these children may be subdivided further

- Octrees work in a similar fashion, but in 3-D, with cubical regions subdivided into 8 parts
Interactive CG Programming: Objectives

- More Sophisticated Interactive Programs
  - Modes of interaction
  - Tools for building
- Techniques
  - Picking: select objects from display (three methods covered)
  - Rubberbanding: interactive drawing of lines, rectangles
  - Display lists: retained mode graphics
Picking [1]:
Definition & Challenges

- Identify User-Defined Object on Display
- In Principle, Should Be Simple
  - Mouse gives position
  - We should be able to determine object-position correspondence
- Practical Difficulties
  - Pipeline architecture: feed forward
  - Hard to map screen back to world
  - Complicated by screen being 2-D, world 3-D
  - How close do we have to come to object to say we selected it?
Picking [2]:
Three Approaches

1. Hit List
   - Most general approach
   - Difficult to implement

2. Buffered Object IDs
   - Write to back buffer or some other buffer
   - Store object IDs as objects rendered

3. Rectangular Maps
   - Easy to implement for many applications
   - e.g., simple paint programs
Rendering Modes

- OpenGL: Can Render in One of Three Modes
  - GL_RENDER
    - Normal rendering to frame buffer
    - Default
  - GL_FEEDBACK
    - Provides list of primitives rendered
    - No output to frame buffer
  - GL_SELECTION
    - Each primitive in view volume generates *hit record*
    - Record placed in *name stack*
    - Stack can be examined later

- **Mode Selected by** `glRenderMode(mode)`
Selection Mode Functions

- `glSelectBuffer()`: Specifies Name Buffer aka Name Stack
- `glInitNames()`: Initializes Name Buffer
- `glPushName(id)`: Push ID on Name Buffer
- `glPopName()`: Pop Top of Name Buffer
- `glLoadName(id)`: Replace Top Name on Buffer
- `id` set by application program to identify objects
OpenGL Functions for Manipulating Name Stack

- `void glInitNames(void);`
  - Creates empty name stack
  - Must call to initialize stack prior to pushing names

- `void glPushName(GLuint name);`
  - Adds name to top of stack
  - Maximum dimension: implementation-dependent
  - Must contain at least 64 names
  - Can query state variable `GL_NAME_STACK_DEPTH`
  - Pushing too many values causes `GL_STACK_OVERFLOW`

- `void glPopName();`
  - Removes name from top of stack
  - Popping value from empty stack causes `GL_STACK_UNDERFLOW`

- `void glLoadName(GLuint name);`
  - Replaces top of stack with name
  - Same as calling `glPopName(); glPushName(name);`

Adapted from tutorial © 2001-2009 A. R. Fernandes
Lighthouse 3D, http://www.lighthouse3d.com
Rendering in Selection Mode: Example

- #define BODY 1
- #define HEAD 2

void renderInSelectionMode()
{
    glInitNames();        // 1. create empty name stack (NS)
    glPushName(BODY);     // 2. push first name
    // 3. hit record (HR) for each primitive intersecting view volume
    drawBody();
    glPopName();          // 4. empty stack & save HRs to selection buffer (SB)
    glPopName();          // 5. new name; no HR, same SB
    drawHead();           // 6. new HR for each primitive in VV
    drawEyes();           // 7. update HR with new max/min depths
    glPopName();          // 8. empty NS; write HRs to SB
    drawGround();         // 9. new HRs; empty NS, depth update only
}

Adapted from tutorial © 2001-2009 A. R. Fernandes
Lighthouse 3D, http://www.lighthouse3d.com
Using Selection Mode

- Initialize Name Buffer *aka* Name Stack
- Enter Selection Mode (using Mouse)
- Render Scene with User-Defined Identifiers
  - Accumulates hits
  - Create new hit record *iff* needed (otherwise update depth)
- Reenter Normal Render Mode
  - Returns number of hits
  - Objects rendered on small area of screen around cursor
- Examine contents of name buffer
  - Hit records written to selection buffer
  - Include information about each hit
    - ID
    - Depth
Selection Mode: Redefining View Volume

- Caveat
  - As just described, selection mode won’t work for picking – why?
  - Because every primitive in view volume will generate a hit
  - Need to change viewing parameters
    - Only those primitives near cursor are in altered view volume
    - Use gluPickMatrix (see Angel 5e or 6e for details)

- New Procedure (cf. Fernandes Tutorial)
  - 1. Get the window coordinates of the mouse
  - 2. Enter selection mode
  - 3. Redefine viewing volume so that only small area of window around cursor is rendered
  - 4. Render scene, either using all primitives or only those relevant to picking operation
  - 5. Exit selection mode and identify objects which were rendered on that small part of screen
Graphical User Interface Design: Using Regions of Screen

- Rectangular Arrangement
  - Used by many applications
  - e.g., paint & computer-aided design (CAD) programs

- Advantages
  - Compared to: selection mode picking
  - Easier to look at cursor position, determine part of window it is in

- Common Graphical User Interface (GUI) Design
Picking:
Using Second Buffer & Color-Coding

- **Color Coding**
  - For small number of objects
  - Can assign a unique color to each object
  - Often assigned in color index mode

- **Using Color Coding for Picking**
  - Render scene to color buffer other than front buffer
  - Results of rendering not visible
  - Get mouse position
  - Use `glReadPixels()` to read color in buffer written at position of cursor
  - Returned color gives ID of object

Adapted from slides © 2005-2008 E. Angel, University of New Mexico
Writing Modes

Application bitwise logical operation

Source \( s \) \( d' \)

read_pixel \( d \)

Destination

write_pixel

frame buffer

Adapted from slides © 2005-2008 E. Angel, University of New Mexico
Exclusive OR (XOR) Write

- Usual (Default) Mode
  - Source replaces destination: $d' = s$
  - Cannot write temporary lines this way – why?
    - Cannot recover what was “under” line in fast, simple way
    - Consequence: cannot deselect (toggle select) easily

- Solution: Exclusive OR Mode (XOR)
  - $d' = d \oplus s$
  - Suppose we use XOR mode to scan convert line $P_0P_1$
  - Can draw it again to erase it!
Rubberbanding

- **Switch to XOR Write Mode**
- **Draw Object**
  - Line
    - Can use first mouse click to fix one endpoint
    - Then use motion callback to continuously update second endpoint
    - Each time mouse is moved, redraw line which erases it
    - Then draw line from fixed first position to new second position
    - At end, switch back to normal drawing mode and draw line
  - Works for other objects
    - Rectangles
    - Circles

“Rubber-Banding with OpenGL”
© 2009 J. Xu
The Code Project
Rubberband Lines: Example

Initial display

Draw line with mouse in XOR mode

Mouse moved to new position

Original line redrawn with XOR

New line drawn with XOR

Adapted from slides © 2005-2008 E. Angel, University of New Mexico
Logical Operations between Two Bits $X$, $Y$

- 2 bits $\Rightarrow 2^2 = 4$ values
- 4 values $\Rightarrow 2^4 = 16$ pairwise functions
  - $X$, $Y$, $\overline{X}$, $X \land Y \equiv XY$, $X \lor Y \equiv X + Y$, $X \oplus Y \equiv X\overline{Y} + \overline{X}Y$
  - etc.
- In general: $2^b$ functions for $b$ bits

All 16 Operations Supported by OpenGL

- Must enable logical operations: `glEnable(GL_COLOR_LOGIC_OP)`
- Choose logical operation
  - `glLogicOp(GL_XOR)`
  - `glLogicOp(GL_COPY)` – default
Immediate versus Retained Modes

- **OpenGL Standard: Immediate Mode Graphics**
  - OpenGL programs use immediate mode by default
  - Once object is rendered, there is no memory of it
  - In order to redisplay it, must re-execute its rendering code
  - Can be especially slow if objects
    - are complex
    - must be sent over network

- **Alternative: Retained Mode Graphics**
  - Accomplished in OpenGL via display lists, vertex buffer objects
  - Define objects
  - Keep them in some form that is easy to redisplay
Display Lists in OpenGL

- Conceptually Similar to Graphics Files
  - Compare: Flexible Vertex Format (FVF) definitions in Direct3D
  - Also compare: mesh formats for OpenGL itself, other CG libraries

- Requirements
  - Define each display list (DL)
    - Name
    - Create
  - Populate: add contents by
    - reading in file
    - generating mesh automatically
  - Close

- Client-Server Environment
  - DL placed on server
  - Can redisplay without sending primitives over network each time
Display List Functions

- Creating Display List
  - GLuint id;
  - void init()
    {
      id = glGenLists( 1 );
      glNewList( id, GL_COMPILE );
      /* other OpenGL routines */
      glEndList();
    }

- Calling Created List
  - void display()
    {
      glCallList(id);
    }

Display Lists & State

  - DLs are syntactic sugar (text abbreviations) for
    - Rendering commands (especially mesh traversal)
    - Parameters
  - Now deprecated! Use vertex buffer objects (VBOs) instead

- Side Effects: State Changes within DLs
  - Most OpenGL functions can be put in display lists
  - State changes made inside DL persist after DL is executed

- Avoiding Unexpected Results
  - Use `glPushAttrib` and `glPushMatrix` upon entering DL
  - Use `glPopAttrib` and `glPopMatrix` before exiting
Hierarchy & Display Lists

- Consider: Model of Car
  - Similar hierarchy to that for general scene graphs
  - Describes relative modelview transformation (MVT)
    - translation
    - rotation (relative Euler angle or quaternion)

- Need to Create Display Lists
  - Chassis
  - Wheel

```c
glCallList( CHASSIS );
glTranslatef( ... );
glCallList( WHEEL );
glTranslatef( ... );
glCallList( WHEEL );
...
glEndList(); NewList( CAR, GL_COMPILE );
```
Picking in Action

FarmVille © 2009 – 2011 Zynga, Inc.
Summary

- Reading for Last Class: §2.4.3, 8.1, Eberly 2e, GL handout
- Reading for Today: Chapter 6, Esp. §6.1, Eberly 2e
- Reading for Next Class: Chapter 7, §8.4, Eberly 2e
- Last Time: Adaptive Spatial Partitioning
  - Trees: VSD, CSG, BSP
  - Spatial partitioning (SP)
    - Examples: BSP trees, quad/octrees (adaptive); voxels (uniform)
    - Scenes: spatial partitioning vs. boundary representation (B-rep)
- Today: Picking
  - OpenGL modes: rendering (default), feedback, selection
  - Name stack
  - Hit records
  - Rendering in selection mode using selection buffer
  - Color coding of pickable objects
- Next Class: Interaction Handling
Terminology

- **Spatial Partitioning (SP):** Calculating Intersection, Visibility
  - Binary Space Partitioning tree – 2-way decision tree/surface
  - Quadtree – 4-way for 2-D
  - Octree – 8-way for 3-D
- **Volume Graphics aka Volumetric Representation:** Uniform SP (Voxels)
- **Boundary Representation:** Describing Enclosing Surface
  - Meshes
  - Implicit surfaces
  - Sweeps (e.g., sphere-swept volumes: sphere, capsule, lozenge)
- **Picking:** Allowing User to Select Objects in Scene
  - Selection mode: mode when cursor (“mouse”) is active
  - Name stack: last in, first out data structure holding object names
  - Hit records: ID, depth info for intersections with view volume
  - Selection buffer: holds hits, depth (compare: frame/z-buffer)
  - Color coding: using color to represent pickable object ID