Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

Module 1B
Picking

Intersection Testing

- Many application level tasks require intersection tests
  - **Picking** – identifying objects at specified screen positions
    - e.g., selection with an input device such as a mouse click
  - **Collision detection** between objects
    - Moving camera and moving objects
  - **Ray tracing** and **ray casting**
- **Bounding volumes (BV)** add efficiency
  - Complex geometry bound with simple volumes
    - Test intersection of simple BV before more expensive tests against complex object
  - Often use a hierarchy of bounding objects
    - Traverse the object tree or scene graph
      - If no intersection occurs with a BV – then the sub tree can be discarded
  - **View frustum culling** (will discuss in week 5)
    - Tests BV against the view frustum
      - If entirely outside do not need to render the objects within the BV

OpenGL Picking Methods

- Legacy OpenGL supports picking / selection
  - Renders objects into small “pick” window
    - Special picking matrix along with the projection matrix to restrict drawing to a small region of the viewport
      - Example use: Finding objects drawn near the cursor
  - Objects that overlap are returned, along with depth extent
  - Generally implemented on the host processor
    - No graphics acceleration
- Not available in OpenGL ES
- General Procedure for OpenGL Picking

```
Draw scene into framebuffer
Enter selection mode
Specify a pick window
Redraw the scene
    Assign identifiers to objects
    (Contents of framebuffer do not change)
Process the returned hit records
```

Using OpenGL Selection

- Specify the array to be used for returned “hit” records
  - **glSelectBuffer(maxSize, buffer)**
- Enter the selection mode
  - **glRenderMode(GL_SELECT);**
- Initialize the name stack
  - **glInitNames()**
- Define the viewing volume for selection
  - Usually different than the framebuffer view volume (e.g. smaller!)
  - **gluPickMatrix()**
- Draw primitives and manipulate the name stack
  - Assign each primitive an appropriate name / ID
  - **glPushName(), glPopName(), glLoadName()**
- Exit selection mode and process the returned hit records

Example

```
GLuint selectBuffer[512];
glSelectBuffer(512, selectBuf);      // Set up selection buffer
glRenderMode(GL_SELECT);      // Enter selection mode
glInitNames();
glPushName(0);
glPushMatrix();                       // Save the current transformation state
     // Create desired pick volume
     glLoadName(1);
     drawObject1();
     glLoadName(2);
     drawObject2();
     glLoadName(3);
     drawObject3();
     drawObject4();                   // Note that this will share ID=3 with object 3!
glPopMatrix();                        // Restore previous transformation
glFlush();

// Get the hit records (return to render mode)
GLint hits = glRenderMode(GL_RENDER);
processHits(hits, selectBuf);
```

Specifying the Pick Region

- Special utility routine **gluPickMatrix** to modify the current projection matrix

```
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluPickMatrix(x, y, width, height, viewport);
gluPerspective();  // Or glOrtho, glFrustum, gluOrtho2D
    // Draw scene for picking
    // Perform picking
glPopMatrix()

// To create a 5x5 pixel region near the cursor location:
GLint viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);     // Or keep a record of the viewport in code!
gluPickMatrix((GLdouble)x, (GLdouble)(viewport[3]-y), 5.0, 5.0, viewport);

(Note that y is inverted (mouse/screen coordinates vs. OpenGL)
```

Hardware Accelerated Picking

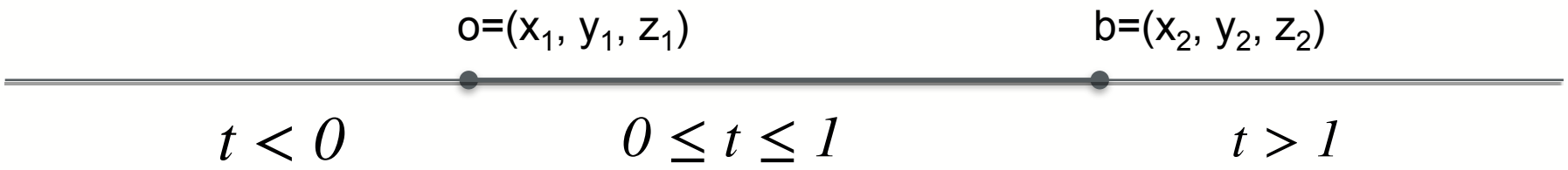
- Section 22.1 (16.1, 3rd Edition)
- Haeberli and Hanrahan suggest rendering scene into z-buffer
  - No lighting
  - Each polygon has different color used as an identifier
  - Color at the pixel location is read to perform the pick
  - Disadvantage: have to render the scene in a separate pass
  - Advantage: more efficient in applications where many pick/ID operations are needed or scene is static
  - Analogous to a 2D picking method where scene is rendered to a bitmap
- Similar method renders identifiers into stencil buffer
  - Many systems use 24 bit Z buffer and 8 bit stencil buffer
    - Stencil values share the same word as depth values
      - Little added cost to set identifiers during same pass as rendering is performed
- Limitation: 8 bits supplies only 256 identifiers
- (Note: we'll see other uses for stencil buffers within this course)

Ray

- Ray defined as an origin point (o) and a direction vector (d)

$$r(t) = \boldsymbol{o} + t\boldsymbol{d}$$

  - Scalar value t, generates different points on the ray
  - t < 0 is behind the ray start point – not part of the ray
- Can create a ray given two points: o=(x_1, y_1, z_1) and b=(x_2, y_2, z_2)
  - d=b-o
  - $x = x_1 + (x_2 - x_1)\, t = x_1 + d_x t$
  - $y = y_1 + (y_2 - y_1)\, t = y_1 + d_y t$
  - $z = z_1 + (z_2 - z_1)\, t = z_1 + d_z t$
  - d_x, d_y, d_z are the **delta** x,y,z
    - Components of d



Distance Along Ray

- If d is a unit vector, then t is **distance** from the origin
- Often only concerned with intersections in **front** of the ray
- Often want to find the closest intersecting object
  - Maintain a **current distance** along the ray
    - To the nearest intersection found
    - Initialize to infinity (MAX_FLOAT)
  - Can ignore any intersections further than the current distance
    - As we intersect “nearer” objects we update the current distance
    - Can allow some algorithm efficiency enhancements