

EECE 478

Rasterization
& Scenes

Learning Objectives

Rasterization
- Be able to describe the complete graphics pipeline.
- Describe the process of rasterization for triangles and lines.

Compositing
- Manipulate alpha blending values for smoothing, compositing and antialisaing.

Learning Objectives

Scenes and Optimization
- Be able to create data structures for entire scenes.
- Be able to evaluate the relative expense of basic rendering sequences.
- Be able to use basic techniques to optimize pipeline rendering.
- Describe and implement various scene culling techniques.

Vocabulary

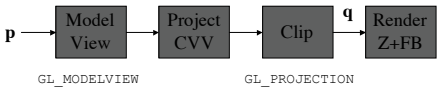
- Rasterization
- Device coordinates
- Clipping
- Bounding box
- Axis-aligned bounding box
- Bounding volume
- Bounding sphere
- Culling
- Back-face culling Tesselation
- Antialiasing
- Compositing
- Alpha blending
- Scene graph

Rasterization

Rasterization: Converting 2D objects to raster or coloured pixels

– End of pipeline ⇒ frame buffer

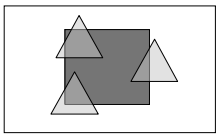
– After clipping against canonical view volume



Clipping

Canonical view volume determines what will be drawn on screen

- Throw away everything outside of CVV
- Must draw *partial* objects at edges



Clipping

Canonical view volume creates *normalized device coordinates*

- x,y,z from [-1,1]

⇒ window coordinates – viewport transform

⇒ screen coordinates – window transform

– convert from float ⇒ integer

Liang-Barsky Clipping

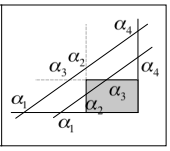
Clipping lines (2D):

- Parametric form
- Order of α_1, α_2, α_3, α_4 determines clip

$$\mathbf{p}(\alpha) = (1-\alpha)\mathbf{p}_1 + \alpha\mathbf{p}_2$$

Solve for α_n such that $\mathbf{p}$ equals

$$\begin{bmatrix}-1\\ y\end{bmatrix}, \begin{bmatrix}x\\ -1\end{bmatrix}, \begin{bmatrix}1\\ y\end{bmatrix}, \text{ and } \begin{bmatrix}x\\ 1\end{bmatrix}$$



Clipping Triangles

Clipping triangle against CVV can produce

- *nothing (culled)*
- triangle, quadrilateral, or pentagon

Optimize tests that can reduce total computation

- try to find *nothing* first

View Volume Culling

Bounding box is a rectangle that minimally encloses the triangle
- parallel to x,y axes ⇒ *axis aligned*
- simple to construct
  - min and max of x,y coords of vertices
- if b-box doesn't intersect CVV, then triangle can't!

$$(\bar{x} > -1) \wedge (\underline{x} < 1) \wedge (\bar{y} > -1) \wedge (\underline{y} < 1)$$

General Volume Culling

A *bounding volume* completely contains some subset of scene
- no part of contents intersects region of interest if BV doesn't
- find bounding volumes such that we can quickly *reject* intersection of bounding volume with ROI

- *Avoid need for complex intersection calculation!*
- Useful for frustum culling, collision detection, …

Line Rasterization

Bresenham's Algorithm: $(x_1,y_1) \rightarrow (x_2,y_2)$

1. For $\Delta x = x_2\text{-}x_1$ $\Delta y = y_2\text{-}y_1$ where $\Delta x > \Delta y$
2. Set $d = 0$ $y = y_1$ $x = x_1$
3. Until $x = x_2$
   1. Plot (x,y)
   2. If $\Delta x > 2(d+\Delta y)$ then
      - $d = d+\Delta y$
   3. Else
      - $d = d+\Delta y\text{-}\Delta x$, $y = y+1$
   4. $x = x+1$

Polygon Rasterization

Start with polygon in device coordinates

Achieve four goals:

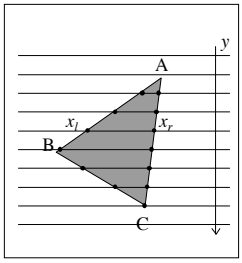
- Projection from 3D to 2D
- Hidden-surface removal
- Shading and colouring
- Rasterization of polygon

End with polygon pixels in frame buffer

Rasterization Strategy

Scan Conversion

- Given *color* and *z* for A, B, C
- Sort A,B,C by y
- Compute x_l & x_r for each y
- Interpolate *color* and *z* for left and right
- Interpolate along each scan line



Compositing

Compositing: Combining a number of images into one

- Can be done at rasterization
- Need some way to control combination
- Easiest form is to control *transparency*
- Often dependent on order of drawing sequence

Billboarding, Transparency, Antialiasing

Alpha Blending

Alpha: Colors have fourth component

- α controls combination of colors with framebuffer contents (per pixel)
- α is *opacity* (α=1 is *fully opaque*)
- (1-α) is *transparency*

$$C_d = \alpha C_s + (1-\alpha) C_d$$

15

OpenGL: Alpha Blending

- Three component colors RGB ⇒ α=1
- Alpha blending must be *enabled*
- Options for blending styles

```
/* Set up for standard Alpha blend */
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA,
             GL_ONE_MINUS_SRC_ALPHA);
```

16

Transparent Objects

Observations:

- Occluded by opaque objects
- Do not occlude opaque objects

Implementation:

- Draw all transparent objects after opaque
- Turn off update of depth buffer
- Retain depth buffer test!

```
/* Prepare to draw translucent object */
glDepthMask (GL_FALSE);
```

17

Object Models

Object Model: Data structure to enable storage, manipulation and rendering

- Divided into objects
  - Any part of model that moves as a unit
  - Articulation allows whole or parts to be moved together maintaining relationships
- Organized for rendering and culling

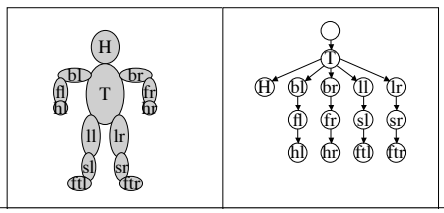
Hierarchical Models

- Hierarchy = tree
  - Root node
  - Every node has ≥0 children
  - Geometry, materials at nodes
  - Transforms on links
  - All objects *below* node are affected by state at node
- Common models can be repeated
  - Directed acyclic graph (DAG)

Example Hierarchy

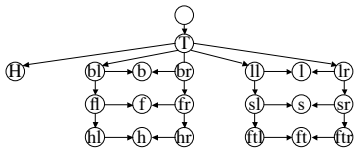
Consider a person

- Each node has geometry
- Each link has instance transform



Shared Geometry

Whenever same shape used
- Point to same geometry (DAG)
- Instance transform allows different sizes



21

Render A Node

Depth-first graph traversal

1. At node:
   a. Bind material
   b. Render geometry
2. For each child link
   a. Push link transform
   b. Recursively render node at link
   c. Pop link transform

22

Tree Data Structure: C

```
typedef struct {
  Material *material;
  Geometry *geometry;
  Child *child;
} Node;
```

```
typedef struct {
  Child *next;
  GLfloat *mat;
  Node *node;
} Child;
```

```
void renderNode (Node *n, State *s) {
  if (n->material) { renderMat (n->material, s); }
  if (n->geometry) { renderGeom (n->geometry, s); }
  for (Child *ch = n->child; ch; ch = ch->next) {
    if (ch->mat) { glPushMatrix();
      glMultMatrix (ch->mat); }
    renderNode (ch->n, s);
    if (ch->mat) { glPopMatrix (); }
}
```

23

Tree Data Structure: C++

```
class Node {                class Child {
  void push (Path *p);        Child *next;
  void pop  (Path *p);        Node *node;
  Child *child;             };
};

void Node::render (Path *p) {
  push (p);
  for (Child *ch = n->child; ch; ch = ch->next) {
    ch->n->render (p->extend (ch->n));
  }
  pop (p);
}
```


Hierarchical Culling(1)

- Each node has *bounding volume*
  - Encloses objects at and below it in hierarchy
  - Computed bottom-up from hierarchy
    - Derive geometric bounding volume directly
    - Transform children by *inverse* of link transform
    - Bounding volume is *union* of transformed children
- Change in transform or geometry must be propagated up hierarchy!
  - Best to keep shallow hierarchies

Hierarchical Culling(2)

Descend through tree

Before rendering a node:

1. Check bounding volume vs. frustum
2. If outside frustum, don't render it or children
3. If entirely inside frustum, turn off checking for children

Occlusion Culling

Occlusion culling: Don't draw primitives that will be occluded by others

- Binary Space Partition (BSP) Trees
  - Recursively partition subspaces by planes
- Portals
  - Indoors with walls and windows, doorways
- Horizon Culling
  - Height fields

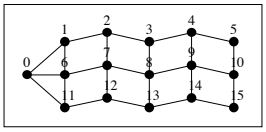
Display Lists

- *Display List*: Fixed, optimized sequence of OpenGL calls.
  - `glGenLists(n)`: Creates *n* display lists
  - `glNewList(i,t)`: Compiles and optionally displays (depending on *t*) list *i*
  - `glEndList()`: Finishes list started by `glNewList`
  - `glCallList(i)`: Execute list *i*

Representing Geometry

Surface representation:

- Set of vertices, colors, normals
- Mesh of primitives using vertices
- Many vertices shared between primitives
  - Can store primitives using indices of vertices

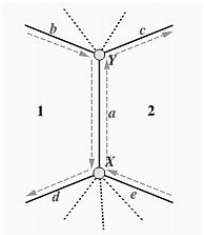


Vertex Arrays

1. Define and activate particular arrays
   - Vertex, color, normal, texture coordinate
   - `glEnableClientState`
   - `glVertexPointer`, `glColorPointer`, …
2. Call drawing primitives that take arrays of indices as arguments
   - `glDrawArrays`: Fixed index order
   - `glDrawElements`: Arbitray index order

Geometric Modeling

Winged Edge:
- Array of Edges



Edge Data Structure: a
- Vertices:
  - Start: X End: Y
- Faces:
  - Left: 1 Right: 2
- Traversals:
  - Left: b, d
  - Right: e, c