Basics of 3D Rendering

CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

http://www.pling.org.uk/cs/cgv.html
What We Have So Far

3D geometry

2D pipeline
Handy Fact

Matrices preserve flat geometry
Handy Fact

Matrices preserve flat geometry
So What?

3D triangles look like 2D triangles under camera transformations.
So What?

3D triangles look like 2D triangles under camera transformations.

Use 2D pipeline for 3D Rendering!
Side Note

Only true for flat shapes

Side Note

Only true for flat shapes

Frame Buffering
Double and Triple Buffering

Tearing: Data from multiple frames appear on the screen at the same time. This happens when GPU rendering rate and monitor refresh rate are not synced.

http://www.newcluster.com/wp-content/uploads/2015/01/g-sync_diagram_0.jpgitokgxy9kpos
Double Buffering with V-Sync

SwapBuffer: Either copy Back Buffer to Front Buffer, or Swap Pointers (Usually in Fullscreen mode).
Double Buffering with V-Sync

SwapBuffer: Either copy Back Buffer to Front Buffer, or Swap Pointers (Usually in Fullscreen mode).

Rendering rate limited by refresh rate 😞
Triple Buffering with V-Sync

Front Buffer

Back Buffer 1

Back Buffer 2

Front Buffer (Video Memory)

Vertical Retraces

Triple-buffering
Triple Buffering with V-Sync

Back Buffer 1
Back Buffer 2
Front Buffer
Extra Video Memory
Occulusion
Painter’s Algorithm

Draw items one at a time
Painter’s Algorithm

Draw items one at a time
Painter’s Algorithm

Draw items one at a time
Painter’s Algorithm

Draw items one at a time
Painter’s Algorithm

Draw items one at a time

Works for most 2D applications, e.g. windowing
What Order?
What Order?

Depth is changing
Early Hidden Surface Approaches

• Pre-compute rendering order
• Cut geometry as needed
• ...

A Characterization of Ten Hidden-Surface Algorithms

IVAN E. SUTHERLAND*, ROBERT F. SPROULL**, AND ROBERT A. SCHUMACKER*

This paper discusses the hidden-surface problem from the point of view of sorting. The various surfaces of an object to be shown in hidden-surface...
Observation

Each pixel can decide what is on top independently.
Observation

Each pixel can decide what is on top independently.

Ed Catmull

Wolfgang Strasser
Color Buffer (RGB each cell)  Depth buffer (one number each cell)
Z-Buffer
Z-Buffer Issues: Resolution

\[ z_{\text{ndc}} = \frac{\text{far} + \text{near}}{\text{far} - \text{near}} + \frac{2 \cdot \text{far} \cdot \text{near}}{(\text{far} - \text{near}) z_{\text{world}}} \]

Non linear!
Z-Buffer Issues: Resolution

\[ z_{\text{ndc}} = \frac{\text{far} + \text{near}}{\text{far} - \text{near}} + \frac{2 \cdot \text{far} \cdot \text{near}}{(\text{far} - \text{near}) \, z_{\text{world}}} \]

\[ \frac{dz_{\text{ndc}}}{dz_{\text{world}}} \propto \frac{1}{z_{\text{world}}^2} \]

Depth
Z-Buffer Issues: Depth Fighting

aka. “Stiching” or “bleeding”

Z-Buffer Issues: Depth Fighting

Hack: Scale and add offset
“glPolygonOffset”

aka. “Stiching” or “bleeding”

Cull [kuhl]:

*To identify and throw away invisible geometry to save processing time.*
Basic Culling Strategies

- **Backface culling**: remove geometry facing away from the camera
- **View volume culling**: remove geometry outside frustum
- **Occlusion culling**: remove invisible geometry
Backface Culling

None  Backface culling  Hidden surface removal

http://medialab.di.unipi.it/web/IUM/Waterloo/node70.html
Specifying Triangle Orientation

```c
glDisable/glEnable(GL_CULL_FACE)
```

**Default:**

```c
glFrontFace(GL_CCW)
```
View Volume Culling

Potential strategies:

• Store scene hierarchically
  • With bounding volumes

• Compute viewing frustrum
  • Don’t render volumes that are clearly outside frustrum
Occlusion Culling: Portal Rendering

http://www.aaid.ca/flash/media/hkmh/images/floor1/000a-geology-portal-cg-rendering.jpg
Occlusion Culling: Portal Rendering

Potentially Visible Set

Occlusion Culling: Portal Rendering

Potentially Visible Set

Occlusion Culling: Portal Rendering

Potentially Visible Set (PVS)


Mirrors ?
Summary of Culling Techniques

- **View Frustum**
- **Backface**
- **Portal**
- **Occlusion**

**Detail**
Acceleration Structures
Goal of Acceleration Structures

• Quickly reject objects that are outside the viewing volume
• Query for intersections efficiently
Spatial Hierarchies

Letters correspond to planes (A)
Spatial Hierarchies

Letters correspond to planes (A,B)
Spatial Hierarchies

Letters correspond to planes (A,B,C,D)
Spatial Hierarchies: Variations

kd-tree  oct-tree  bsp-tree
Octree

• Each node has 0 or 8 children
  • Each node can equally subdivide its space (an AABB) into eight subboxes by 3 midplanes
  • Children of a node are contained within the box of the node itself
  • Stop subdividing when number of objects/primitives falls below a threshold or maximum depth has reached.

• Recursively render cells that intersects with the viewing volume
K-d Tree

- Begin with the global bounding box containing all primitives.
- Choose an axis and a splitting plane perpendicular to that axis.
- Subdivide the primitives on both sides of the plane into two groups.
  - Usually done in a balanced manner.
- Stop when the number of primitives in each single group is below a threshold.
Shadows
Shadows
Shadows: Spatial Cue

http://mamassian.free.fr/papers/mamassian_tics98.pdf
Shadows: Realism

Shadow

- Shading
- Attached shadow
- Inter-reflection
- Cast shadow
- Penumbra
We will only concentrate on hard-shadows
Shadow Mapping

• First Pass
  • Render the Scene from the light Source
    • Pretend the light is the “camera”
  • Store the depth buffer as a texture
    • Heightfield – tells us the “distance” of the nearest points from the light source.

Light’s POV depth map
Shadow Mapping

• Second Pass
  • Project the depth buffer texture from the light’s P.O.V
  • Render the scene from the camera position

Recall Projective Texturing
Projective Texturing (RECALL)

• Map NDC (-1, 1) to Texture Coordinate space (0-1)
  • Scale and add Bias

\[
\begin{bmatrix}
    s'' \\
    t'' \\
    r'' \\
    q''
\end{bmatrix}_{\text{TextureSpace}} =
\begin{bmatrix}
    0.5 & 0 & 0 & 0.5 \\
    0 & 0.5 & 0 & 0.5 \\
    0 & 0 & 0.5 & 0.5 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    s' \\
    t' \\
    r' \\
    q'
\end{bmatrix}_{\text{NDC}}
\]

Final texture coordinates after perspective-correct interpolation of \((s'', t'', r'', q'')\)

\[
\left( \frac{s''}{q''}, \frac{t''}{q''}, \frac{r''}{q''} \right)
\]

Compare this with depth
Shadow Mapping

• Second Pass
  • Project the depth buffer texture from the light’s P.O.V
  • Render the scene from the camera position
  • Compare fragment’s depth (projected r texture coordinate) to the depth stored in texture

```glsl
depthMapValue = texture( depthTexture, projCoords.st / projCoords.q );
fragmentDepth = (projCoords.r / projCoords.q);
float shadow = fragmentDepth > depthMapValue ? 1 : 0;
```
Shadow Mapping: Issues

• Limited field of view of depth map
Shadow Mapping: Issues

• Limited field of view of depth map
• Z-Fighting
  • Add scale and bias – similar to glPolygonOffset
  • Getting it right is complicated
Shadow Mapping: Issues

• Limited field of view of depth map
• Z-Fighting
  • Add scale and bias – similar to glPolygonOffset
  • Getting it right is complicated
• Sampling problem (aliasing)
  • Larger depth map may mitigate some of it
Shadow Mapping: Issues

• Limited field of view of depth map
• Z-Fighting
  • Add scale and bias – similar to glPolygonOffset
  • Getting it right is complicated
• Sampling problem
  • Larger depth map may mitigate some of it

Lots of paper about soft shadows
Deferred Rendering

a.k.a Deferred Shading
Deferred Rendering

• So far: we did **Forward Rendering**
• Lots of fragments are wasted due to overdraw
  • Complex Lighting/Shading computation wasted
• Solution: “Defer” lighting computation until we have figured out all the pixels that end up on the screen
• Deferred Rendering can handle lots of lights
• Complexity:
  • Forward Rendering: Num_Objects * Num_Light
  • Deferred Rendering: Num_Object + Num_Light

http://learnopengl.com/#!Advanced-Lighting/Deferred-Shading
Deferred Rendering

Two Pass
1. Geometry Pass
2. Lighting Pass

http://learnopengl.com/#!Advanced-Lighting/Deferred-Shading
Deferred Rendering

Two Pass
1. Geometry Pass
2. Lighting Pass

http://learnopengl.com/#!Advanced-Lighting/Deferred-Shading
Deferred Rendering

• Transparency still done through Forward Rendering
  • Need to copy the Depth Buffer.

http://learnopengl.com/#!Advanced-Lighting/Deferred-Shading
Deferred Rendering: Lots of Light

• Can handle lots of light: key is **Light Volume**
  • Shade pixels that are close to a light
  • Why does not “if-else” branch work for this on the GPU?

http://learnopengl.com/#!Advanced-Lighting/Deferred-Shading
Deferred Rendering: Lots of Light

• Can handle lots of light: key is **Light Volume**
  • Shade pixels that are close to a light
  • Why does not “if-else” branch work for this on the GPU?
• Draw one light volume at a time: accumulate colors

[Image of a 3D scene with two spheres, one wireframe and one shaded]

http://learnopengl.com/#!Advanced-Lighting/Deferred-Shading
Deferred Rendering: Lots of Light

- Can handle lots of light: key is Light Volume
- Shade pixels that are close to a light
- Why does not “if-else” branch work for this on the GPU?
- Draw one light volume at a time: accumulate colors

http://learnopengl.com/#!Advanced-Lighting/Deferred-Shading
Deferred Rendering: Challenges

• Doesn’t support MSAA (Multiple Sample Anti-Aliasing)
• Extra frame buffer memory
• Transparencies need to be done with Forward Rendering
Challenges of Rasterizers
Shadows and Reflections
Transparencies

http://www.archicadwiki.com/Bugs/TransparencyIn3dWindow
Depth of Field

Basics of 3D Rendering

CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

http://www.pling.org.uk/cs/cgv.html