ECE 4893A/CS 4803 MPG: Multicore and GPU Programming for Video Games

Lecture 6: 3D Rendering Pipeline (III)

Prof. Hsien-Hsin Sean Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
• Paint each pixel’s color (by calculating light intensity) on your display (Typically object-based)
• Gouraud Shading:
  – Intensity Interpolation of vertices
Gouraud Shading

- Scan conversion algorithm
Comparison of Shading Methods

- Phong shading
  - requires generating per-pixel normals to compute light intensity for each pixel, not efficient for games
  - Can be done on GPGPU today using Cg or HLSL
- Gouraud shading is supported by Graphics hardware

Source: Michal Necasek
Double Buffering

- Display refreshes at 60 ~ 75 Hz
- Rendering could be “faster” than the refresh period
- Too fast leads to
  - Frames not shown
- Too slow leads to
  - New and old frame mixed
  - Flickering
- Solution:
  - Double or multiple buffering
Z-Buffer

- Also called depth buffer
- Draw the pixel which is nearest to the viewer
- Number of the entries corresponding to the screen resolution (e.g. 1024x768 should have a 768k-entry Z-buffer)
- Granularity matters
  - 8-bit never used
  - 16-bit z value could generate artifacts
Z-Buffer Bandwidth Could Be an Issue

- Perform Z test before drawing a pixel
- How much bandwidth needed for RZ+WZ+RC+WC +TR per pixel?

- Overdrawn rate is about 3 out of 4
  - One every 4 pixels drawn is for clearing the Z-buffer
- Some cards perform Z-compression to alleviate bandwidth issues
Aliasing

- Jagged line (or staircase)
- Can be improved by increasing resolution (i.e. more pixels)
Anti-Aliasing by Multisampling
(Example: Supersampling)

- GPU samples multiple locations for a pixel
- Several different methods
  - e.g., grid (as shown), random, GeForce’s quincunx
- Downside
  - Blurry image
  - Increased memory (e.g., z-buffer) storage for subpixel information

Example

3x3 Virtual Pixels
(Bartlett window)

Actual Screen Pixels

(255, 159, 159)

(255,255,255) (255,255,255) (255,0,0)

(255,255,255) (255,0,0) (255,255,255)

(255,0,0) (255,255,255) (255,255,255)

(255,0,0)
Anti-Aliasing Example

Ideal

No MSAA

With MSAA
Visualizing Anti-Aliasing Example

No MSAA

With MSAA
Texture Mapping

• Rendering tiny triangles is slow
• Players won’t even look at some certain details
  – Sky, clouds, walls, terrain, wood patterns, etc.
• Simple way to add details and enhance realism
• Use 2D images to map polygons
• Images are composed of 2D “texels”
• Can be used to substitute or blend the lit color of a texture-mapped surface
Texture Mapping

• Introduce one more component to geometry
  – Position coordinates
  – Normal vector
  – Color
  – Texture coordinates

• Texture info
  – \((u, v)\) coordinates for each vertex
  – Typically range from 0 to 1
    • \((0,0)\) from upper left corner
Texture Mapping

Texture: hunk.jpg

\{(v1.x, v1.y, v1.z, \ldots, 1, 0),
{v2.x, v2.y, v2.z, \ldots, 1, 1),
{v0.x, v0.y, v0.z, \ldots, 0, 0),
{v3.x, v3.y, v3.z, \ldots, 0, 1),}
Texture Mapping

Texture 1
(0,0) (0,1)

Texture 2
(1,0) (1,1)
Texture Mapping

\{(v1.x, v1.y, v1.z, \ldots, 0, 0),
   (v2.x, v2.y, v2.z, \ldots, 5, 0),
   (v0.x, v0.y, v0.z, \ldots, 5, 3),
   (v3.x, v3.y, v3.z, \ldots, 0, 3)\}
Texture Mapping

\[
\begin{align*}
\{v1.x, v1.y, v1.z, \ldots, 0, 0\}, \\
\{v2.x, v2.y, v2.z, \ldots, 6, 0\}, \\
\{v0.x, v0.y, v0.z, \ldots, 6, 6\}, \\
\{v3.x, v3.y, v3.z, \ldots, 0, 6\},
\end{align*}
\]
Magnification

- Texel and pixel mapping is rarely 1-to-1
- Mapped triangle is very close to the camera
- One texel maps to multiple pixels
Nearest Point Sampling (for Magnification)

- Choose the texel nearest the pixel’s center
Bi-linear Filtering (for Magnification)

- Average for the 2x2 texels surrounding a given pixel
Minification

- Texel and pixel mapping is rarely 1-to-1
- Multiple texels map to one pixel
Nearest Point Sampling (for Minification)

- Choose the texel nearest the pixel’s center
Bi-linear Filtering (for Minification)

- Average for the 2x2 texels surrounding a given pixel
Mip-mapping

• Multiple versions are provided for the same texture
• Different versions have different levels of details
  – E.g., 7 LOD maps: 256x256, 128x128, 64x64, 32x32, 16x16, 8x8, 4x4
  – Choose the closest maps to render a surface
• Maps can be automatically generated by 3D API

• Accelerate texture mapping for far-away polygons
• More space to store texture maps
Mip-mapping

- API or Hardware can
  - Generate (lower resolution) mip maps automatically
  - Choose the right one for the viewer
    - Good performance for far triangles
    - Good LOD for close-by objects
  - Tri-linearly interpolate
Tri-linear Filtering using Mip maps

• Interpolate between mipmaps
Anisotropic Filtering

- Not isotropic
- Preserving details for oblique viewing angles (non-uniform surface)
- AF calculates the “shape” of the surface before mapping
- The number of pixels sampled depends on the distance and view angles relative to the screen
- Very expensive

Source: nvidia
Color Blending and Alpha Blending

- Transparency effect (e.g. Water, glasses, etc.)
- Source color blended with destination color
- Several blending methods
  - Additive
    \[ C = \text{SrcPixel} \otimes (1,1,1,1) + \text{DstPixel} \otimes (1,1,1,1) = \text{SrcPixel} + \text{DstPixel} \]
  - Subtractive
    \[ C = \text{SrcPixel} \otimes (1,1,1,1) - \text{DstPixel} \otimes (1,1,1,1) = \text{SrcPixel} - \text{DstPixel} \]
  - Multiplicative
    \[ C = \text{DstPixel} \otimes \text{SrcPixel} \]
  - Using Alpha value in the color (Alpha Blending)
    \[ C = \text{SrcPixel} \otimes (\alpha,\alpha,\alpha,\alpha) + \text{DstPixel} \otimes (1-\alpha,1-\alpha,1-\alpha,1-\alpha) \]
  - And many more in the API ...
Alpha Blending (Inverse Source form)

No transparency

Src=0.2 (triangle)  
Dest=0.8 (square)

Src=0.5 (triangle)  
Dest=0.5 (square)

Src=0.8 (triangle)  
Dest=0.2 (square)
Another Example Without Transparency
Another Alpha Blending Example

Src=0.3 (rect) Dest=0.7 (checker)
Src=0.5 (orange rect) Dest=0.5
Src=0.6 (Triangle) Dest=0.4
Alpha Test

- Reject pixels by checking their alpha values
- Model fences, chicken wires, etc.

Texture: bar.jpg

if (\(\alpha \ op \ val\))
  reject pixel
else
  accept pixel
Multi-Texturing

• Map multiple texture to a polygon
  – Common APIs support 8 textures
• Performance will be reduced
• Multiple texturing stages in the pipeline
• Texture color will be calculated by
  – Multiplication
  – Addition
  – Subtraction
Multi-Texturing Example: Light Mapping

- Some crumpled paper texture
- A spotlight map
- Different alpha blending
Bump Mapping

- **Per-fragment lighting** using bump map (normal map) to perturb surface normal
- No geometry tessellation, avoid geometry complexity
- Store normal vectors rather than RGB colors for bump map

Source: wikipedia

Shaded sphere  Bump map  Bump mapped sphere
Bump Mapping

- Normal map was derived from a Height field map
  - Height field stores the “elevation” for each texel
  - Sample texel’s height as well as texels to the right and above
Environment Mapping

- Cube Map Textures
- Each face encodes 1/6 of the panoramic environment

Source: Addison-Wesley (Cg Tutorial)
Reflective Environment Mapping

- Look up the environment map
- Add a reflection to a fragment’s final color

Source: zabur.smz.sk
Stenciling

• Stencil buffer
  – To reject certain pixels to be displayed
  – To create special effect similar to alpha test
    • Mask out part of the screen
  – Set together with Z-buffer in 3D API
  – Perform prior to Z-buffer test

\[
\text{if } ((\text{stencil ref} \& \text{ mask}) \text{ op } (\text{pixel val} \& \text{ mask})) \\
\text{ accept pixel} \\
\text{else} \\
\text{ reject pixel}
\]
Stencil Buffer Example

This window area is set to be drawn by the stencil buffer.

Reject pixels inside this area.
Apply Fog Effect

- Provide depth cue
  - Simulate weather condition
  - Avoid *popping* effect
- Color blending

\[
\text{color} = (1 - f) \cdot \text{Color}_{\text{vertex}} + f \cdot \text{Color}_{\text{fog}}
\]

\[
f = \frac{\text{dist(eye,vertex)} - f \text{og}_{\text{start}}}{f \text{og}_{\text{top}} - f \text{og}_{\text{start}}}
\]

- Calculate distance
- Calculate intensity of vertex color based on distance
  - Color blending
  - Linear density, exponential density
- Blending color schemes
  - Per-vertex (then Interpolate pixels), less expensive
  - Per-fragment basis (Nvidia hardware), better quality