Fragment Operations

CPSC 414

The Rendering Pipeline

Geometry Processing

Geometry Database → Model/View Transform → Lighting → Perspective Transform. → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer

Rasterization → Fragment Processing

Scan Conversion

So far:
- Know
  - how to determine which pixels are covered by primitive (transformations, scan-conversion)
  - Which pixels are not occluded (depth test)
  - what color they should have (lighting, shading)

Now:
- Combine resulting pixels/fragments with current framebuffer
  - Alpha and stencil test
  - Color blending

Pixels vs. Fragments

Pixel:
- Color of an image at one point (defined before)

Fragment:
- Information about one point of a geometric primitive
  - Color, depth, texture coordinates,…
- Associated with a particular pixel in the framebuffer (i.e. we know where it is going to end up), but NOT stored in the framebuffer, yet!
- Scan-conversion generates individual fragments for the geometric primitives
  - Then handed from stage to stage of rendering pipeline

Alpha and Stencil Test

Synopsis:
- Depth test is not the only way in which fragments can be terminated (removed from pipeline) without actually drawing them
- Alpha test: perform simple test on alpha channel of incoming fragment
  - i.e. test of incoming fragment
- Stencil test: perform simple test on the value of an additional buffer in the video memory
  - i.e. test of current state in video RAM!

Alpha Test

Alpha:
- An additional color channel for every color (i.e. in addition to red, green, and blue)
- Does NOT correspond to any visible color
- Can be used for multiple purposes
  - Transparency is just one
- Specification in OpenGL:
  - glColor4f( red, green, blue, alpha );
  - All channels between 0 and 1
  - In the pipeline, all fragments have an alpha value!
  - Default value (for glColor3f): 1

© Wolfgang Heidrich
University of British Columbia
Alpha Test

Operations:
• Compare alpha value of incoming fragment to a constant
  – Terminate all fragments that do not pass the test
  – Comparison operators:
    ‧ <, <=, = , >= , >, ≠
    ‧ E.g.: if ( fragment.Alpha < constant )…
  – In addition:
    ‧ always (default), never

Applications:
• Masking out parts of the geometry
  • E.g. rectangle with RGBA texture on it

Stencil Test

Stencil Buffer:
• An additional buffer in video memory
• Provides one channel (typically 1..8 bit) for every pixel

Operations on Stencil Buffer:
• Counting value for one pixel up or down for every fragment drawn
• Set stencil value to constant if fragment is drawn
• Clear/initialize value for all pixels

Tests that can be performed:
• Same as for alpha (<, <=,...)
• But remember: this tests content of video RAM, not value of incoming fragment!

Example (Rendering of Decals):
• Decals: small polygons co-planar with a large one that simulate painting etc. on the large polygon
  • E.g.: street demarkations
  • Algorithm:
    – Clear stencil buffer to 0
    – Draw decal polygons, set stencil to 1 for all pixels drawn
    – Draw street polygon only where stencil = 0
**Relation of Depth, Alpha, Stencil Test**

- **Alpha Test**
  - terminate
- **Stencil Test**
  - terminate, possibly update stencil value
- **Depth Test**
  - terminate, possibly update stencil value

**The Rendering Pipeline**

1. **Geometry Database**
2. **Model/View Transform.**
3. **Lighting**
4. **Perspective Transform.**
5. **Clipping**
6. **Scan Conversion**
7. **Texturing**
8. **Depth Test**
9. **Blending**
10. **Frame-buffer**

**Blending**

**Synopsis:**
- Combine fragment color ("source color") and framebuffer color ("destination color") using a weighted sum
  \[ C_d = a \cdot C_s + b \cdot C_d \]
- \( a \) and \( b \) can be chosen from a number of possibilities:
  - Constants 0 and 1
  - Source or destination color
  - Source or destination alpha
  - 1- any of the above

**Blending Example: Partially transparent surfaces**
- Use alpha channel of object color to encode transparency
  - E.g. alpha = 0.1 means we see what is behind the object with 10% of the intensity, and the object itself with 90% intensity
  - Thus:
    - \( a = 1 - \) source alpha
    - \( b = \) source alpha

**Transparent Surfaces**

**Problem:**
- Have to make sure that everything located behind the transparent object has been drawn already once the transparent object is rendered
- Thus:
  - Draw all opaque objects first
  - Then draw transparent objects in back-to-front order
  - Depth sorting of geometry required!