The Graphics Pipeline

Topics

1. Defining graphics architectures
2. The graphics pipeline
3. Computation and bandwidth requirements
Papers

Required
1. The design of the OpenGL graphics interface, M. Segal and K. Akeley

Optional
1. The OpenGL Specification, M. Segal and K. Akeley

Both papers will be available from course web site:
http://www.graphics.stanford.edu/courses/cs448a-01-fall

Lecture slides will also be available online

Graphics Systems and Libraries

Declarative (What, not How)
Describe the scene
For example: virtual camera
Systems
- RenderMan scene description
- Inventor and Performer scene graphs

Imperative (How, not What)
Emit a sequence of drawing commands
For example: load model-view matrix
Systems
- OpenGL
- PostScript and Xlib
Drawing Commands

```c
# glBegin(GL_POLYGON);
# glColor(RED);
# glVertex3i(0,0,0);
# glVertex3i(1,0,0);
# glVertex3i(0,1,0);
# glEnd()
```

```c
# glBegin(GL_POLYGON);
# glColor(RED);
# glVertex3i(0,0,0);
# glColor(BLUE);
# glVertex3i(1,0,0);
# glColor(BLUE);
# glVertex3i(0,1,0);
# glEnd()
```

OpenGL Architecture

```
Image

Unpack Pixels -> Pixel Operations

Texture Memory

Point. Line. Polygon Rasterization

Frame Buffer

Geometry

Unpack Vertexes -> Vertex Operations

Fragment Operations
```

```
Image Rasterization

Pack Pixels

Unpack Pixels

Pixel Operations

Fragment Operations

Frame Buffer
```

CS448 Lecture 2
Kurt Akeley, Pat Hanrahan, Fall 2001
API = ISA

VLIW-like instructions

- Specify operations
  - Multiple functional units
  - Orthogonal operations
- Specify data paths
  - Composition of operations
ISA Specification

Invariance

- Does NOT precisely define drawing commands
  For example:
  - Does not specify what pixels are inside a triangle
  - Does not specify the precision of intermediate calculations
- Image drawn by two systems may differ
- Does require invariance across modes
  - Image drawn in two modes must be the "same"

Modern Graphics Pipeline

Application
  ↓
Command
  ↓
Geometry
  ↓
Rasterization
  ↓
Texture
  ↓
Fragment
  ↓
Display

Forward-Algorithm
A trip down the graphics pipeline
**Application**

Simulation
Input event handlers
Modify data structures
Database traversal
Primitive generation
Utility functions

**Command**

Command buffering
Command interpretation
Unpack and perform format conversion

**Maintain graphics state**

```c
glLoadIdentity();
glMultMatrix(T);
glBegin(GL_TRIANGLE_STRIP);
glColor3f(0.0, 0.5, 0.0);
glVertex3f(0.0, 0.0, 0.0);
glColor3f(0.5, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glColor3f(0.0, 0.5, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glMultMatrix(T);
glColor3f(0.5, 0.0, 0.0);
glVertex3f(1.0, 1.0, 0.0);
... glEnd();
```
Geometry

- Evaluation of polynomials for curved surfaces
- Transform and projection
- Clipping, culling and primitive assembly

Object-space triangles

Screen-space lit triangles

Texture coordinate generation

Lighting (light sources and surface reflection)
**Rasterization**

Setup (per-triangle)

Sampling (triangle = \{fragments\})

Interpolation (interpolate colors and coordinates)

---

**Texture**

Texture transformation and projection

Texture address calculation

Texture filtering
Fragment

Texture combiners

Texture Fragments

Fragments

Textured Fragments

Fragment

Texture combiners and fog
Owner, scissor, depth, alpha and stencil tests
Blending or compositing
Dithering and logical operations

Textured Fragments

Framebuffer Pixels
Display

Gamma correction
Analog to digital conversion

Framebuffer Pixels ➔ Light

Graphics Pipeline

Application ➔ Command ➔ Geometry ➔ Rasterization ➔ Texture ➔ Fragment ➔ Display

Application ➔ Object ➔ Rasterization ➔ Image ➔ Display
Graphics Pipeline

Functionality vs. Frequency

Geometry processing = per-vertex
- Transformation and Lighting (T & L)
- Floating point; complex operations
- 10 million vertices

Fragment processing = per-fragment
- Blending and texture combination
- Fixed point; limited operations
- 1 billion fragments
Evolution of the Graphics Pipeline

History

- Framebuffers: display
- Geometry processing: transformation and lighting
- Rasterization: hidden surface and simple shading
- Texturing: perspective correct texture lookup
- Antialiasing: multisampling
- Shading: multiple textures and texture combiners

Where and how to insert new functionality?

Inserting Functionality: Order

Hidden-surface elimination

- painters algorithm = hide-first
- z-buffer = hide-last

Texturing

- Fragment textures = texture-last
- Vertex textures = texture-first

Shading

- Vertex shading = shade-first or shade-first-vertex
- Fragment shading = shade-last-fragment
- Deferred shading = shade-last-pixel
Reyes Architecture

Application → Command → Tessellation → Shade/Texture → Sample → Fragment → Display

Application → Object → Sample → Image → Display

Computational Requirements

Application → Command → Geometry → Rasterization → Texture → Fragment → Display

Geometry (per-vertex)
Assumptions:
- 1 infinite light
- Texture coordinates

<table>
<thead>
<tr>
<th>Operation</th>
<th>ADD</th>
<th>CMP</th>
<th>MUL</th>
<th>DIV</th>
<th>SPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Ops</td>
<td>40</td>
<td>8</td>
<td>53</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Rough estimate: 100 ops
Computational Requirements

Rasterization: per-vertex
Assumptions:
- 7 interpolants (z,r,g,b,s,t,q)

Rough estimate

<table>
<thead>
<tr>
<th>ADD</th>
<th>CMP</th>
<th>MUL</th>
<th>DIV</th>
<th>SPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>22</td>
<td>55</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Computational Requirements

Rasterization: per-fragment
Assumptions:
- 7 interpolants (z,r,g,b,s,t,q)

Rough estimate

<table>
<thead>
<tr>
<th>ADD</th>
<th>CMP</th>
<th>MUL</th>
<th>DIV</th>
<th>SPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Computational Requirements

Application

Command

Geometry

Rasterization

Texture

Fragment

Display

Texture: per-fragment

Assumptions:
- Projective texture mapping
- Level of detail calculation
- Trilinear interpolation

<table>
<thead>
<tr>
<th>ADD</th>
<th>CMP</th>
<th>MUL</th>
<th>DIV</th>
<th>SPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>5</td>
<td>48</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Rough estimate

Computational Requirements

Application

Command

Geometry

Rasterization

Texture

Fragment

Display

Fragment: per-fragment

Assumptions:
- Texture blending
- Color blending
- Depth buffering

<table>
<thead>
<tr>
<th>ADD</th>
<th>CMP</th>
<th>MUL</th>
<th>DIV</th>
<th>SPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Rough estimate
Computational Requirements

<table>
<thead>
<tr>
<th>Application</th>
<th>Per-Vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td></td>
</tr>
<tr>
<td>Geometry</td>
<td></td>
</tr>
<tr>
<td>Rasterization</td>
<td></td>
</tr>
<tr>
<td>Texture</td>
<td></td>
</tr>
<tr>
<td>Fragment</td>
<td></td>
</tr>
<tr>
<td>Display</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Per-Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Texture Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 GB/s</td>
</tr>
</tbody>
</table>

FrameBuffer

<table>
<thead>
<tr>
<th>Framebuffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 GB/s</td>
</tr>
</tbody>
</table>

Communication Requirements

<table>
<thead>
<tr>
<th>Application</th>
<th>Rough estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>5 Gops</td>
</tr>
<tr>
<td>Geometry</td>
<td>150 Gops</td>
</tr>
<tr>
<td>Rasterization</td>
<td></td>
</tr>
<tr>
<td>Texture</td>
<td>4 GB/s</td>
</tr>
<tr>
<td>Fragment</td>
<td>0.36 GB/s</td>
</tr>
<tr>
<td>Display</td>
<td></td>
</tr>
</tbody>
</table>

0.880 GB/s
Graphics State or Context

*Required to minimize data transmission*

Resources (shared or global, persistent)
- Fonts
- Texture
- Display lists

Attributes
- Appearance: Lights, Materials, Colors, ...
- Transformation: camera, model, texture, ...
- Options: fb formats, constant per-frame

Graphics State

Ideally small and bounded
- e.g. maximum number of lights
  - OpenGL: ~12kb

Distributed throughout the pipeline
  - Difficult to manage (forces major design decisions)
  - Must often be broadcast; must be consistent

Hard to switch contexts
  - OpenGL has a single context; X has multiple contexts
  - One drawing process; windows are difficult

Difficult to query state