Geometry

Real-Time
Graphics Architecture

Kurt Akeley
Pat Hanrahan

http://graphics.stanford.edu/courses/cs448-07-spring/

Outline

Vertex and primitive operations
SGI implementations
Reyes architecture
Primitive generation (geometry shader)
**Readings**

**Required**

**Recommended**
- *OpenGL Specification*
- *Direct3D 10 Specification*

---

**Modern Graphics Pipeline**

- **Application**
  - **Command**
  - **Geometry**
  - **Rasterization**
  - **Texture**
  - **Fragment**
  - **Display**

*Today’s lecture*
*Already covered*
Geometry Processing

Two types of operations

- Vertex operations
  - Operate on individual vertexes
- Primitive operations
  - Operate on all the vertexes of a primitive

Engineering Challenges

Correctness

- Invariance
- Consistency (e.g. frame to frame)

Parallelism

- Dynamic load balancing due to variable amount of work (rasterization, clipping, generation, ...)
- Ordering constraints in the graphics pipeline

Managing state
Vertex Stage

Vertex Operations

Transform coordinates and normal
- Model → world (not rigid)
- World → eye (rigid)

Normalize the length of the normal

Compute vertex lighting

Transform texture coordinates
- Generate if so specified

Transform coordinates to clip coordinates (projection)

Divide coordinates by w

Apply affine viewport transform (x, y, and z)
Primitive Stage

Primitive Operations

Primitive assembly
Clipping
Backface cull

Transform coordinates and normal
  - Model \(\rightarrow\) world
  - World \(\rightarrow\) eye
Normalize the length of the normal
Compute vertex lighting
Transform texture coordinates
Transform to clip coordinates
Assemble vertexes into primitives
Clip primitives against frustum
Divide by \(w\)
Apply affine viewport transform
Eliminate back-facing triangles
**Primitive Assembly**

Assemble based on application commands
- Independent or strip or mesh or ....

Decompose polygons/quads into triangles
- Prior to clipping to maintain invariance

Algorithm properties
- Fixed execution time (good)
  - All vertex operations up to this point have this property
- Independent vertexes become independent primitives (bad)

---

**Clipping**

Two types
- Point, line: eliminates geometry
- Polygon: eliminates and introduces edges

---

Line Segments

![Diagram showing line segments and a polygon](image-url)
Clipping

Two types
- Point, line: eliminates geometry
- Polygon: eliminates and introduces edges

Invariance requirements
- Pre-decomposition to triangles
- Canonical order of vertexes for edge calc.

Algorithm properties
- Vertex interdependencies (bad)
- Data-dependent execution (worse)
  - Variable execution time (substantially different)
  - Variable code paths

Backface Cull

Facet facing toward or away from viewpoint?
- No facet normal (other APIs?)
- Use sign of primitive's window coordinate “area”
  - Remember, only triangles are planar

Use facing direction to
- Select lighting result (for n or -n)
- Potentially discard the primitive

Move earlier to avoid work?

Triangle area \(= \frac{(x0y1 - x1y0) + (x1y2 - x2y1) + (x2y0 - x0y2)}{2}\)
SGI Generations

Engineering Challenges

- Correctness
  - Invariance
  - Consistency
- Parallelism
  - Dynamic load balancing
  - Ordering constraints in the graphics pipeline
Some Examples

Systems

- Clark Geometry Engine (1983)
- Silicon Graphics GTX (1988)
- Silicon Graphics RealityEngine (1992)
- Modern GPU (2001)

What we’ll look at

- Organization of the geometry system
- Distribution of vertex and primitive operations
- How clipping affects the implementation

Clark Geometry Engine (1983)

Simple, fixed-function pipeline

- Twelve identical engines
- Implemented dot4
- Soft-configured at start-up

Clipping allocated ½ of total ‘power’

- Performance invariant (good)
- Typically idle (bad)
GTX Geometry Engine (1988)

Variable functionality pipeline
- 5 identical engines
- Modes alter some functions
  - Load balancing is difficult
Clipping allocated 1/5 of ‘power’
- Clip testing is always done
- Actual clipping is seldom done
  - Slow pipeline execution (bad)
    out of balance
  - Typically isn’t invoked (good)

RealityEngine Geometry (1992)

Geometry processor
- Eight identical engines
- Handles strips of length n vertexes
- Variable-functionality MIMD organization; Why?
  - Handles clipping
  - Strips of different lengths
  - Different types of primitives
Command processor
- Splits large strips into n vertex strips
- Introduces redundant work (2 extra vertexes)
- Good ‘static’ load balancing
- Shadows per-vertex attribute state
- Sends all attributes with each vertex
- Broadcasts other state
RealityEngine Clipping

Clipping introduces data-dependent (dynamic) load
- Cannot be predicted by CP

Dynamic load balance accomplished by:
- Round-robin assignment
- Large input and output FIFOs
  - For each geometry processor
  - Sized greater than \((n)\) long/typical
- Large work load per processor
  - Minimize the long/typical ratio
  - Unlike pipeline processing

InfiniteReality Geometry (1996)

Geometry processor
- Variable-functionality MIMD organization
- Four identical (SIMD) engines
- Handles strips of length \(n\)

Command processor
- Splits large strips of primitives
- Least-busy work assignment
- Good ‘static’ load balancing
- Shadows per-vertex attribute state
- Sends all attributes with each vertex
- Broadcasts other state

Aggregation
- Reorder output using sequence tokens
InfiniteReality Clipping

Dynamic load balance accomplished by:

- Least-busy assignment
- Even larger FIFOs
- Large work load per processor
- Sequence token to reorder

Likelihood of clipping reduced by

- Guard-band clipping algorithm

Guard-Band Clipping Example

- Rendered
- Discarded
- Clipped
Guard-Band Clipping

Expand clipping frustum beyond desired viewport
- Near and far clipping are unchanged
- Frustum clip only when necessary

Ideal triangle cases:
- Discard if outside viewport, else
- Render without clipping if inside frustum, else
  - Rasterizer must scissor well for this to be efficient
- Clip triangles
  - That cross both viewport and frustum boundaries
  - That cross the near or far clip-planes

Operation is imperfect, but conservative

Modern Geometry Engine

Homogeneous coordinate rasterization algorithm
- Modify inside test
  - Evaluate 3 edges equations
  - Evaluate 6 clipping plane equations\n  - Succeed if inside all
- Fixed amount of work per fragment
- Clipping doesn’t increase or decrease #triangles

Triangle scan conversion using 2D homogenous coordinates, M. Olano, T. Greer, GH1997

Studying this algorithm would be a good project
Primitive Generation Stage

REYES Architecture
D3D 10 Geometry Shader

REYES

Renders Everything You Ever Saw (name by Carpenter)

The Road to Point Reyes
Directed by R. Cook, LucasFilm 1983
Reyes Architecture Reasoning

- Nyquist-sized micro-polygon
  ¼ pixel area (½ pixel size)
  Micro-polygon replaces fragment as the low-level primitive
  One geometry/shading/texturing calculation per micropolygon
  ‘Non-flat’ geometry requires tessellation
  Tessellation free compared to shading

- Shading before hiding
  Hiding=rasterization+z-buffer+csg+transparency+motion+dof
  Displacement maps (texture before sampling/hiding)

- Split and dice along surface parameters (u, v)
  Coherent geometry and texture (filter in texture-space)
**REYES Immediate-Mode Pipeline**

```
Object
- Command
  - Vertex
  - Rasterize

OpenGL
- Texture
  - Fragment
  - Display

REYES
- Command
  - Transform
  - Tessellate

Texture
- Texture
- Shade
- Rasterize
- Display

Image
```

**Primitive Generation (GPUs)**

Why add primitive generation stage?

Some important reasons:
- Provide high-level primitives to applications
- Shift computation from CPU to GPU
- Reduce storage requirements
- Perverse desire to complicate the GPU design ;-

Highest-priority reason:
- Reduce CPU to GPU data rate!
Other Rate Reduction Techniques

Display lists
- Work well for client-server on network
- Works well for CPU-GPU host-graphics memory
- Complexity in managing display lists

Geometric compression and decompression
- Strips and meshes
- Indexed triangle sets
- Vertex buffer objects
- Geometry Compression (Deering '95)

Geometry Primitive Generation Stage

Here?

| Application | Command | Prim. Generate | Vertex | Rasterization | Texture | Fragment | Display |

Or Here?

| Application | Command | Vertex | Prim. Generate | Rasterization | Texture | Fragment | Display |
Geometry Shader

Where to put this stage?
- Requires primitives, thus after primitive assembly
- Do not duplicate transforms on vertexes (for adaptive tessellation), thus after vertex stage
- Assumes shading done in fragment stage
- Assumes texturing can be done in vertex stage for displacement mapping

Input: One triangle or line + neighborhood

Output: 0 to n vertexes defining new primitive
1. Input type may be different than output type
2. May output multiple strips of same output type
Challenges

Enabling parallelism while enforcing order
- Introduces dependency
- Difficult to load balance
  - Variable amount of computation
- Difficult to manage state
  - Variable amount of output

Solution
- Limited size output buffer
  - 1024 32-bit floats per invocation

Problems with Primitive Generation

Cracks and T-vertexes
- Different surfaces must abut
  - But arithmetic is of finite precision
- Generally requires “stitching”
Stitching Patches Together

4x4 Patch 5x5 Patch

Stitching vertexes introduced

CS448 Lecture 7  Kurt Akeley, Pat Hanrahan, Spring 2007
Problems with Primitive Generation

Cracks and T-vertexes
- Different surfaces must abut
  - But arithmetic is of finite precision
- Generally requires “stitching”

Parallel adaptive tessellation is hard (good project)
- Required by REYES algorithm to match tessellation rate with sampling rate

Themes

Correctness
- Invariance
- Consistency

Parallelism
- Dynamic load balancing due to variable work
- Ordering constraints in the graphics pipeline
- Rasterization and generation are complex

Evolution
- Where to add the next stage?