Real-Time Graphics Architecture

Kurt Akeley
Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

Geometry

Outline

- Vertex and primitive operations
- System examples
  - emphasis on clipping
- Primitive generation
- OpenGL selection
Readings

Required


Recommended

- *Curved PN Triangles*, PDF to be on web site soon
- *OpenGL 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

Vertex Operations

Transform coordinates and normal
- Model $\rightarrow$ world (not rigid)
- World $\rightarrow$ 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 \( z \))
Coordinate Transformation

4x4 matrix, 4-component coordinate
Single-precision floating point is typical
Matrices can be composed
  - By the application
  - By the implementation
    - Be careful about invariance

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  m_{11} & m_{12} & m_{13} & m_{14} \\
  m_{21} & m_{22} & m_{23} & m_{24} \\
  m_{31} & m_{32} & m_{33} & m_{34} \\
  m_{41} & m_{42} & m_{43} & m_{44}
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

Normal Transformation

\((n_x' \ n_y' \ n_z') = (n_x \ n_y \ n_z) \ M_u^{-1} \quad M_u = \text{upper-left 3x3 of } M\)

Approaches to acquiring \(M_u^{-1}\) include
  - Maintain separately, or
  - Compute when coordinate matrix is changed, or
  - Force specification by application

Why transform normals? (Can lighting be computed in model coords?)
  - Model matrix may not be rigid
  - But, recall quadrilateral decomps. problem

Why normalize normals to unit length
  - Lighting equations require unit length
  - Non-rigid model matrix distorts lengths
  - Requires reciprocal square root operation
Normal Transformation

\[(n_x', n_y', n_z') = (n_x, n_y, n_z) M_u^{-1} \quad M_u = \text{upper-left 3x3 of } M\]

Approaches to acquiring \(M_u^{-1}\) include
- Maintain separately, or
- Compute when coordinate matrix is changed, or
- Force specification by application

Why transform normals? (Can lighting be computed in model coords?)
- Model matrix may not be rigid
- But, recall quadrilateral decomp. problem

Why normalize normals to unit length
- Lighting equations require unit length
- Non-rigid model matrix distorts lengths
- Requires reciprocal square root operation

Lighting

Simple \(n \cdot l\) evaluation
- Plus ambient and specular
- Multiple lights, ...
- Remember, \(n\) is not a facet normal

Possibly two-sided
- Compute for \(n\) and \(-n\)
- Both results may be needed!
- Much arithmetic is shared

View-direction simplification
- Eye vector is \([0,0,1]\)
- Saves arithmetic

There is no such thing as view direction!
No Interdependencies

Vertex operations apply to vertexes independently

- 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
- Divide by $w$
- Apply affine viewport transform

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 to triangles
- Prior to clipping to maintain invariance

Algorithm properties
- Fixed execution time (good)
  - All vertex operations up to this point have this property
- Vertex interdependencies (bad)

---

**Clipping**

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

---

![Diagram of Line Segments and Polygon](image)
Clipping

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

Invariance requirements
- Pre-decomposition to triangles
- Care with edge arithmetic

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

Advance in sequence to improve efficiency?

Triangle area: \[
\frac{(x_0y_1 - x_1y_0) + (x_1y_2 - x_2y_1) + (x_2y_0 - x_0y_2)}{2}
\]
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)

1st generation capability
Simple, fixed-function pipeline
- Twelve identical engines
- Soft-configured at start-up

Clipping allocated ½ of total ‘power’
- Performance invariant (good)
- Typically idle (bad)
GTX Geometry Engine (1988)

2nd generation capability
Variable functionality pipeline
  - 5 identical engines
  - Modes alter some functions
    - Load balancing is difficult
Clipping allocated 1/5 of ‘power’
  - Clip testing is the constant load
  - Actual clipping
    - Slow pipeline execution (bad)
      - out of balance
    - Typically isn’t invoked (good)

RealityEngine Geometry (1992)

3rd generation capability
Variable-functionality MIMD organization
  - Eight identical engines
  - Round-robin work assignment
  - Good ‘static’ load balancing
Command processor
  - Splits large strips of primitives
  - Shadows per-vertex state
  - Broadcasts other state
Primitive assembly
  - Complicates work distribution
  - Reduces efficiency of strip processing
## 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 \((n) \text{ long/typical}\)
- Large work load per processor
  - Minimize the \(\text{long/typical}\) ratio
  - Unlike pipeline processing

---

## InfiniteReality Geometry (1996)

3\(^{rd}\) generation capability

Variable-functionality MIMD organization
- Four identical (SIMD) engines
- Least-busy work assignment
- Good 'static' load balancing

Command processor
- Splits large strips of primitives
- Shadows per-vertex state
- Broadcasts other state

Primitive assembly
- Complicates work distribution
- Reduces efficiency of strip processing
InfiniteReality Clipping

Dynamic load balance accomplished by:
- Least-busy assignment
- Even larger FIFOs
- Large work load per processor

Likelihood of clipping reduced by
- Guard-band clipping algorithm

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
Guard-Band Clipping Example

- Rendered
- Discarded
- Clipped

Viewport

Modern Geometry Engine (2001)

Utilizes homogeneous rasterization algorithm
- No clipping required
- Hence no primitive assembly prior to rasterization
  - Push backface cull to rasterization setup
  - This is where triangle area is computed anyway

All geometry engine calculations are
- On independent vertexes - easy work distribution
- Not data dependent - minimal code branching

Allows efficient, SIMD geometry engine implementation

Programmability?
Primitive Generation

Why do this?
Some important reasons:
- Match application semantics
- Move computation from CPU to GPU
- Reduce storage requirements
- Perverse desire to complicate the GPU design ;-)  

Highest-priority reason:
- Reduce CPU to GPU data rate!

Basic Idea

Introduce new pipeline stage
- We’ll treat as part of geometry

Two-step application
- Specify generation function
  - Modes
  - Data values (potentially large)
- Execute generation function
  - Block-mode commands, and/or
  - Vertex-like commands

Generated geometry
- Is treated as if app-specified
Problems with Primitive Generation

Specification may be complex
- E.g. 4th order 2D OpenGL Evaluation mesh
- May be more data than generated triangles!
- Specification/execution may be sequential
  - Require double buffer, separate execution engines

Cracks and T-vertexes
- Different surfaces must abut
  - But arithmetic is of finite precision
- Generally requires “stitching”
  - OpenGL accommodates this with mixed evaluation and vertex specification

Stitching Patches Together

4x4 Patch  5x5 Patch
Stitching Patches Together

4x4 Patch  5x5 Patch

Stitching vertexes introduced

Problems with Primitive Generation

Implementation may be complex

- E.g. Trimmed NURBS
  - Limit tesselation to curve-specified region of 2D domain
- Algorithm doesn’t “fit” GPU architecture
  - Don’t redesign a general-purpose MP system

Semantics may not match application’s

- E.g. Trimmed NURBS
- Getting this right is extremely difficult
  - Don’t trespass on application’s “secret sauce”

Where should primitive generation be done? ....
Where is Primitive Generation Done?

On geometry engines
- Reduces maximum vertex rate
  - Generation consumes GE cycles
- May serialize setup/execution
  - GE's not designed to double buffer
- Complicates load balancing
  - Command Processor distribution
- Complicates state handling
  - Command Processor managed this
OpenGL designed to accommodate this
- Awkward state semantics

Where is Primitive Generation Done?

On an additional processor
- Requires fixed-function resources
  - Idle during normal operation
- May serialize setup/execution
  - If not designed to double buffer
- Simplifies
  - Geometry Engine load balancing
  - State management
Feels like a new pipeline stage
- Command processor functions split
Other Data Rate Reductions

Display lists
- Work well for client-server
- Allow bandwidth to be reassigned

Geometric decompression
- Strips and meshes
- Indexed triangle sets
- Geometry Compression (Deering ’95)
- Hypothetical backend for advanced compression format

ATI PN Triangle technology ...

ATI’s TRUFORM Technology

Requires minimal setup
- Single mode (triangle edge subdivision count)
- No geometric data

Operates on triangles as normally submitted
ATI’s TRUFORM Technology

Improves silhouettes
Improves lighting over per-vertex (per fragment?)

Limitations
- Constant subdivision per object
- Generally limited capability (e.g. continuity)

Strengths
- Actually reduces CPU to GPU data rate
- Requires minimal application recoding
- Easily avoids cracks and T-vertexes
- Is simple to understand, to implement, and to use

Leads toward a Reyes rendering approach
- Lots of small triangles
- Shading done in pre-projected coordinates
- Hardware trails software ....
OpenGL Selection

Light pen replacement mechanism
Light pen is a calligraphic device
- Focuses on screen
- Signals when stroke is drawn in focus region
Raster equivalent
- Point with a mouse
- Set "selection mode"
- Re-render entire scene
  - Clip frustum reduced to small region around pointer
  - Each object tagged (integer name)
- Return "hit" information

Real-Time Graphics Architecture

Kurt Akeley
Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall