Real-Time Graphics Architecture

Kurt Akeley

Pat Hanrahan

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

Rasterization

Outline

- Fundamentals
- Examples
- Special topics (Depth-buffer, cracks and holes, …)

Required reading

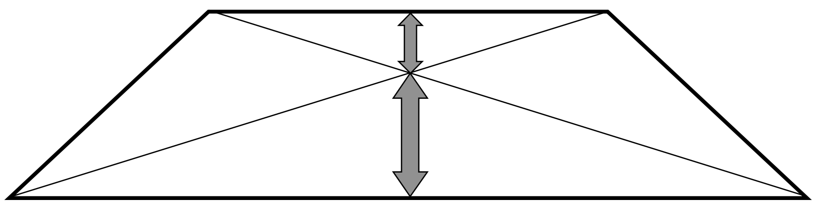
- *Triangle Scan Conversion using 2D Homogeneous Coordinates*, Olano and Greer, EWGH 1997.
- *Optimal Depth Buffer for Low-Cost Graphics Hardware*, Lapidous and Jiao, EWGH 1999.

Recall that …

Straight lines project to straight lines

- When projection is to a plane (our assumption)
- Only vertexes need to be transformed
- That’s why we’re interested in lines and polygons

Projected distance is warped:



Recall that …

Ideal screen coordinates are continuous

- Implementations always use discrete math, but with substantial sub-pixel precision
- A pixel is a big thing
  - Addressable resolution equal to pixels on screen
  - Lots of data (recall over-square RealityEngine buffer)

Points and lines have no geometric area ….

Terminology

Rasterization: convert primitives to fragments

- Primitive: point, line, polygon, glyph, image, ….
- Fragment: transient data structure, e.g.

```
short x,y;
long depth;
short r,g,b,a;
```

Pixels exist in an array (e.g. framebuffer)

- Have implicit <*x*,*y*> coordinates

Fragments are routed to appropriate pixels

- First "sort" we've seen
- There will be more

Two fundamental operations

Fragment selection

- Identify pixels for which fragments are to be generated
- Must be conservative, efficiency matters
- <*x*,*y*> parameters are special

Parameter assignment

- Assign parameter values to each fragment
- E.g. color, depth, …

Fragment selection

Generate one fragment for each pixel that is intersected by the primitive

Intersected could mean primitive's area intersects:

- The square pixel region, or
- The pixel's filter function, or
- The pixel's center point

All three meanings are useful:

- Box sample: tiled rasterization
- Filter function: antialiased rasterization
- Point sample: standard aliased rasterization

Fragment selection (continued)

What if the primitive doesn't have an area? (Points and lines don't.)

- Rule-based approach (e.g. Bresenham line), or
  - Allows desired properties to be maintained, but
  - May require additional hardware complexity
- Assign an area (e.g. circle for point, rectangle for line)
  - Can utilize polygon rasterization algorithm, but
  - May result in wavy lines, flashing points, etc.

Note: point sample and box sample differ

- Cannot simply scale the primitive

Parameter assignment

Identify a parameter function (height-above-plane)

Sample this function as required

Which function?

- Lots of possibilities (that we will ignore)
- Always defined implicitly by vertex values
  - Linear in either screen or object space

Properties of vertex-defined function:

- Zero-order continuity
- Triangles allow the surface to be a plane
- Polygons (4+ edges) are almost never planar
  - Variant with screen orientation?

Linear interpolation

Compute intermediate parameter value

- Along a line: $P = aP_1 + bP_2$, $a+b=1$
- On a plane: $P = aP_1 + bP_2 + cP_3$, $a+b+c=1$

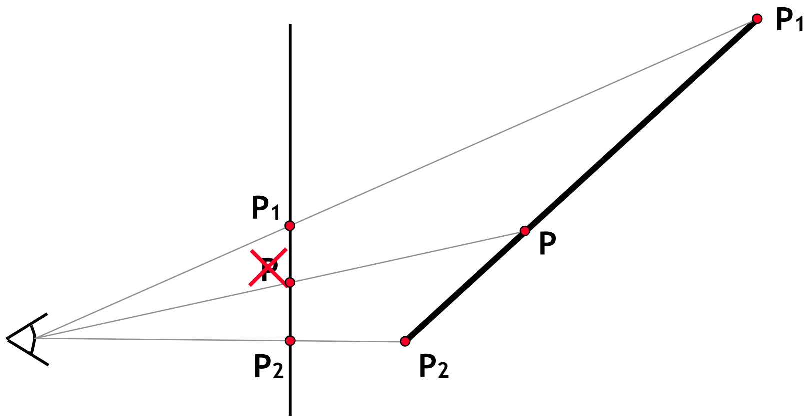
Only projected values interpolate linearly in screen space (straight lines project to straight lines)

- x and y are projected (divided by w)
- Parameter values are not naturally projected

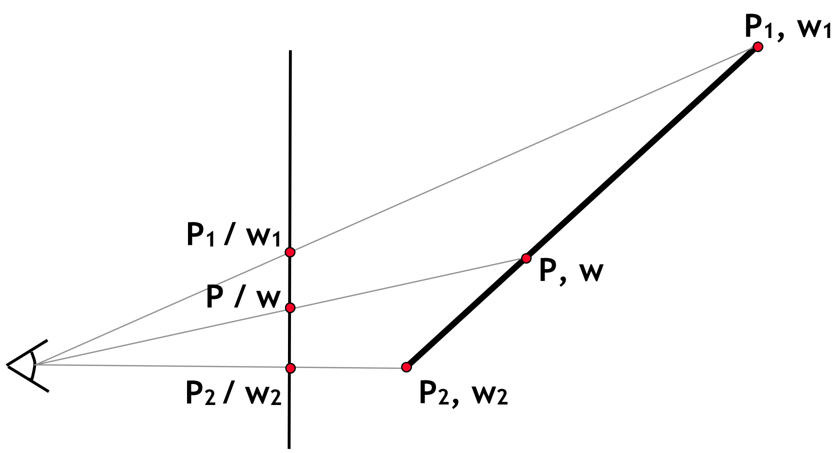
Choice for parameter interpolation in screen space

- Interpolate unprojected values
  - Cheap and easy to do, but
  - Gives wrong values (sometimes OK for color, though)
  - Texture coordinates can't be interpolated this way
- Do it right (next slides)

Projection to straight lines



Projection to straight lines



Interpolate, then project = project, then interpolate

Perspective-correct linear interpolation

Linearly interpolate *P/w* and *1/w*

- Both are projected, so project to straight lines
- (Interpolate→project = project→interpolate)

At the desired sample point

- Recover *P* by dividing *P/w* by *1/w*
- Division is expensive, so
  - Recover w for the sample point (reciprocate), and
  - Multiply each projected parameter value by *w*

$$P = \frac{aP_1/w_1 + bP_2/w_2 + cP_3/w_3}{a/w_1 + b/w_2 + c/w_3} \qquad a + b + c = 1$$

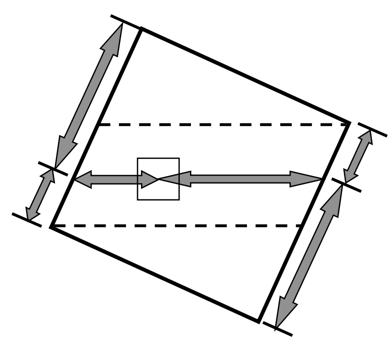
Example: Gouraud shaded quadrilateral

Fragment selection

- Walk edges
- Change at vertexes

Parameter assignment

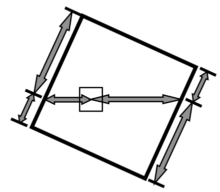
- Two-stage
  - Interpolate along edges
  - Interpolate edge-to-edge
- Three distinct regions
  - Loop is complex
  - E.g. 2/3 regions
- Function of
  - Screen orientation
  - Choice of ⇔ ⇕ spans



Example: Gouraud shaded quadrilateral

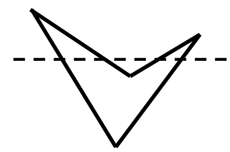
"All" projected quadrilaterals are non-planar

- Due to discrete coordinate precision



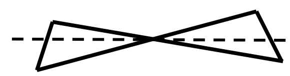
What if quadrilateral is concave?

- Concave is complex (split spans -- see example)
- Non-planar → concave for some view



What if quadrilateral intersects itself?

- A real mess (no vertex to signal change -- see example)
- Non-planar → "bowtie" for some view



All polygons are triangles (or should be)

Three points define a plane

- Can treat all triangles as planar
- Can treat all parameter surfaces as planar

Triangle is always convex

- Regardless of arithmetic precision
- Simple rasterization, no special cases

Modern GPUs decompose *n*-gons to triangles

- SGI switched in 1990, VGX product
- Optimal quadrilateral decomposition invented

Normal-based quad decomposition

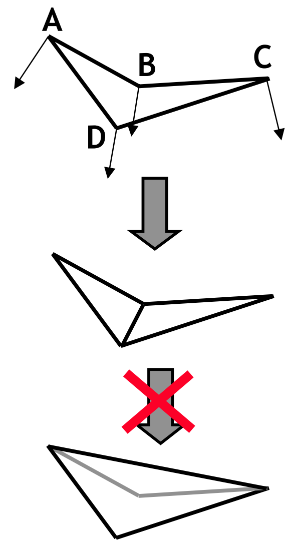
Compute *(A dot C)* and (*B dot D)*

Connect vertex pair with the greater dot product

- Avoid connecting the stirrups

Must avoid frame-to-frame jitter

- Cannot transform normals, or
- Planar quads will jitter



Point sampled triangles

Modern choice for aliased rendering

Fragment selection

- Include if center point is inside
- Handle edge/vertex intersections

Parameter assignment

- Sample function at pixel center
- Mean value for surrounded pixels
- Consistent ray for depth buffer

Never sample outside the triangle

- Avoid color wrap
- But how is antialiased filtering handled?

Point sampled points and lines

Points and lines have no area, so

- Pixel sample locations almost never "in" primitive
- Semantics are confused at best

Must assign "area" parameter functions

- Point: parameter-constant disk
- Line: single-parameter-slope rectangle

Problem: how to outline filled, depth-buffered triangles?

- Depth values are "wrong", lines disappear
- VGX introduced "hollow polygons"
- OpenGL 1.1 introduced glPolygonOffset()

Integer DDA arithmetic

Goal: efficient interpolation

Direct evaluation is expensive

- Requires multiplications for each evaluation

Digital Differential Analyzer (DDA)

- Fixed point *iiiiii.ffffff* representation, accumulator and slope
- Add slope repeatedly to accumulator to evaluate adjacent sample locations
- Planar DDA uses separate *X* and *Y* slopes
  - Can move around the plane arbitrarily
- Require log2(*n*) fraction bits for *n* accumulation steps

Triangle Rasterization Examples

Gouraud shaded (GTX)

Per-pixel evaluation (Pixel Planes 4)

Edge walk, planar parameter (VGX)

Barycentric direct evaluation (InfiniteReality)

Small tiles (Bali – proposed)

Homogeneous recursive descent (NVIDIA)

Algorithm properties

Setup and execution cost

- Absolute
- Relative

Ability to parallelize

Ability to cull to a rectangular screen region

- To support tiling
- To support "scissoring"

Gouraud shaded (GTX)

Two stage algorithm

- DDA edge walk
  - fragment selection
  - parameter assignment
- DDA scan-line walk
  - parameter assignment only

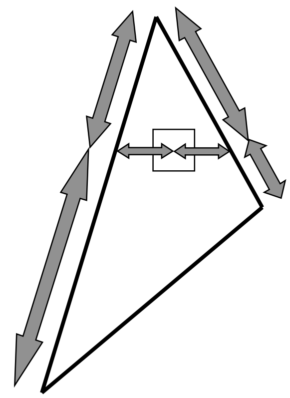
Requires expensive scan-line setup

- Location of first sample is non-unit distance from edge

Parallelizes in two stages (e.g. GTX)

Cannot scissor efficiently

Works on quadrilaterals



Engine-per-pixel (Pixel Planes 4)

Sorry, no diagram ☹

Individual engine at each pixel

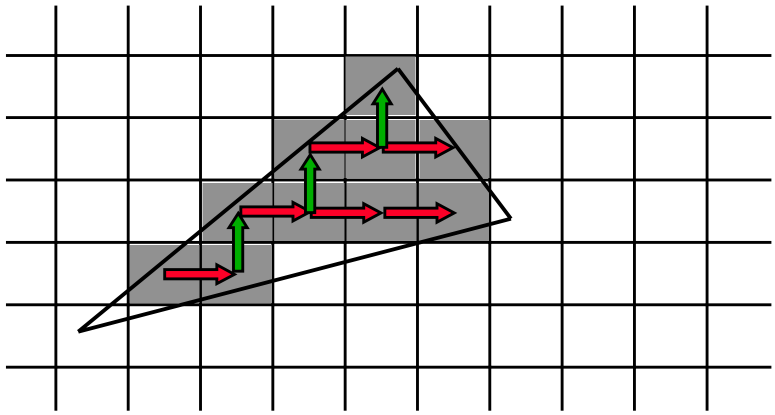
- Solves edge equations to determine inclusion
- Solves parameter equations to determine values

Setup involves computation of plane and edge slopes

Execution is

- Extremely fast (all pixels in parallel)
- Extremely inefficient for small triangles
  - Pixel depth complexity = # triangles in scene
  - Scissor culling is a non-issue

Edge walk, planar evaluation (VGX)



CS448 Lecture 6 Kurt Akeley, Pat Hanrahan, Fall 2001

Edge walk, planar evaluation (VGX)

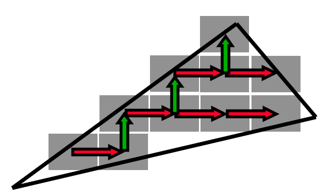
Hybrid algorithm

- Edge DDA walk for fragment selection
  - Efficient generation of conservative fragment set
- Sample DDA walk for parameter assignment
  - Never step off sample grid, so
  - Never have to make sub-pixel adjustment

Scissor cull possible

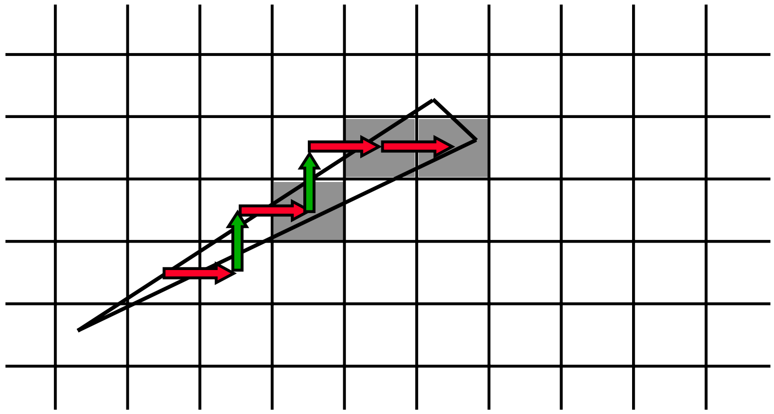
- Adds complexity to edge walk

Sample walk simplifies parallelism



CS448 Lecture 6 Kurt Akeley, Pat Hanrahan, Fall 2001

Interpolation outside the triangle



CS448 Lecture 6 Kurt Akeley, Pat Hanrahan, Fall 2001

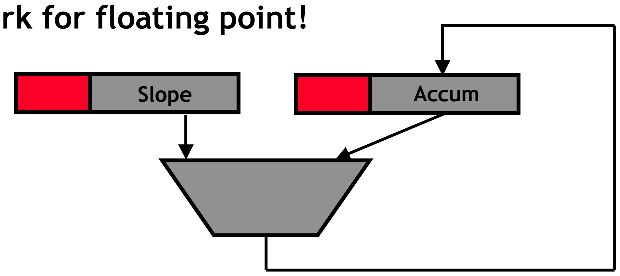
DDA can operate out-of-range

MSBs beyond desired range don't matter

- Carry chain flows up, not down
- Can handle arbitrarily large slopes
- Can iterate outside the triangle's area

Don't clamp intermediate results!

Doesn't work for floating point!



CS448 Lecture 6 Kurt Akeley, Pat Hanrahan, Fall 2001

Guard bits

Problem: overflow or underflow of accumulated value

- Integer arithmetic "wraps"
  - Maximum value overflows to zero
  - Zero underflows to maximum value
- Minor accumulation error → huge value error

Use guard bit(s) to avoid wrapping

- Maintain one or more extra MSBs throughout
- Split guard range equally above and below

Guard bits (continued)

"value"	Guard Bit	Integer part	Clamped
5	1	01	3 (11)
4	1	00	3 (11)
3	0	11	3 (11)
2	0	10	2 (10)
1	0	01	1 (01)
0	0	00	0 (00)
-1 (6)	1	11	0 (00)
-2 (7)	1	10	0 (00)

```
if (guard bit is '0')
    return value;
else if (MSB of value is '1')
    return 0;
else
    return maximum value;
```

DDA bit assignment examples

Edge equation in 4k x 4k rendering space

- 1 guard bit
- 12 integer bits (4k space)
- 10 sub-pixel position bits
- 12 interpolation bits (log2(4096))
- 35 bits total

Depth value in 4k x 4k rendering space

- 2 guard bits (depth wrap is disaster)
- 24 integer bits (reasonable depth precision)
- 13 interpolation bits (longest path in 4k x 4k)
- 39 bits total

Barycentric (InfiniteReality)

Hybrid algorithm

- Approximate edge walk for fragment selection
  - Pineda edge functions used to generate AA masks
- Direct barycentric evaluation for parameter assignment
  - Barycentric coordinates are DDA walked on grid
  - Minimizes setup cost
  - Additional computational complexity accepted
  - Handles small triangles well

Scissor cull implemented

- Supports "guard band clipping"

Small tiles (Bali – proposed)

Framebuffer tiled into *nxn* (16x16) regions

- Each tile is owned by a separate engine

Two separate rasterizations

- Tile selection (avoid broadcast, conservative)
- Fragment selection and parameter assignment

Parallelizes well

Handles small triangles well

Scissors well

- At tile selection stage

Homogeneous recursive descent

Rasterizes unprojected, unclipped geometry

- Huge improvement for geometry processing!
- Interpolates clip-plane distances

Modern choice for GPUs

- But is not well documented
- Read Olano and Greer
  - Parameter assignment precision has many pitfalls
  - Watch out for infinities!

Recursive descent

- Scissors well
- Drives *nxn* (2x2) parallel fragment generation

Cannot generate perspective-incorrect parameter values

Ideal depth buffer

Topic is fractal

- What is good metric for accuracy?

"Ideal" depth buffer (true object-Z buffer)

- Interpolate *Z/w* and *1/w*
- Divide at each fragment to recover object *Z* value
- Expensive division arithmetic (*Z* has lots of bits)
- Precision is distributed evenly in *Z* buffer
  - Seems desirable, but actually is not
  - More precision nearer the viewpoint is good
- SGI "Odyssey" product is only example I know

1/w depth buffer (aka W-buffer)

Observe that *w* is just object *Z* to begin with

Recall that *1/w* interpolates linearly

Store and compare 1/*w* values in depth buffer

No expensive division is required

No additional interpolation, *1/w* was needed anyway

W-buffer precision is packed toward the view point

- Derivative of *1/w* is $-1/w^2$
- Some warp is desired, but this is extreme
- Precision between view point and near-clip is lost
  - *1/w* value is scaled to match far-clip, but not biased

Becoming commonly used

Z/w depth buffer (aka Z-buffer)

Z/w interpolates linearly too

Store and compare *Z/w* values in depth value

No expensive division required

Depth buffer precision is packed toward the near clipping plane (not view point)

- Similar warp to W-buffer
- Lose farclip/nearclip bits in far field
- All precision available near-clip to far-clip

OpenGL/SGI standard approach

- See OpenGL spec for details

Depth buffer warp compensation

Use floating point representation for depth values

May convert to float after interpolation if desired

To compensate for warp in W-buffer or Z-buffer

- Set far to 0.0, near to maximum value
- InfiniteReality product does this (Z-buffer)

To compensate for warp in object-Z buffer

- Set far to maximum value, near to 0.0
- Odyssey product does this

Depth buffer x,y precision

Depth parameter "surface" is constructed from vertex values, not from first principles

- Discretized *x*,*y* move this surface substantially
- Could compute depth plane with high-precision *x*,*y*, but
- This path leads to sampling outside the triangle

A 32-bit depth buffer in a system with 2-bit subpixel precision makes no sense!

Holes and cracks

Assume point sampled triangles

Goal for adjacent triangles:

- No missed pixels (holes)
- No pixels rasterized twice

Problem: sample point intersects the edge

- Use canonical edge arithmetic (identical for both triangles)
- Swap only the sense of the decision

Problem: sample point intersects shared vertex

- Construct elaborate rasterization rules, or
- Use separate grids for vertexes and samples

Holes and cracks (continued)

T-vertex

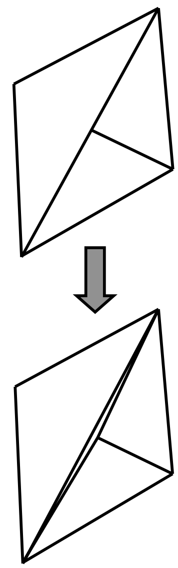
- A vertex intended to be "on" an edge
- Results in cracks (see example)

Cannot eliminate problem

- Would require infinite precision
- Antialiasing helps, though

See Pat Hanrahan's cs248 slides

- graphics.stanford.edu/courses/cs248-98-fall/Lectures/lecture9/



Final observations

To get consistent precision using floating point

- Add a bias equal to maximum value
- Forces all exponents to be the same
- Can subtract or lose in float-to-fixed conversion

Round-to-zero is evil for signed rasterization algorithms

- Especially important for floor() and ceiling()
- Or avoid with bias technique

Some horrible problems go away in the limit

- Huge parameter slope → little triangle area

Final observations (continued)

Highly-acute triangles

- Result from high-precision screen coordinates
- Can rasterize incorrectly due to minor slope errors
- Develop very large parameter slopes

Sometimes excess precision is damaging

- Screen coordinates → acute triangles
  - Olano and Greer describe this
- But, screen coordinates → depth buffer accuracy

Line stipple is a mess

- For one segment, complicates raster parallelism
- For connected segments, complicates geometry parallelism

Real-Time Graphics Architecture

Kurt Akeley

Pat Hanrahan

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