COS 344: L7 Chapter 9: The Graphics Pipeline

Cobus Redelinghuys

University of Pretoria

11/03/2024

Chapter 9: Introduction

- Ray tracing is part of image-order rendering.
- This chapter will look at object-order rendering.
- Ray tracing is easier, as no "complicated" mathematics is required to render a simple scene.
  - But is inefficient, as each pixel is considered in turn and objects are repeatedly visited.
- Instead, the object-order rendering considers each geometric object in turn and finds the pixels that the object could affect.

Rasterization

The process of finding all the pixels in an image that are occupied by a geometric primitive.

- Object-order rendering is more efficient, due to only visiting each geometric object once.

The graphics pipeline outline will be used for the remainder of Chapter 9.

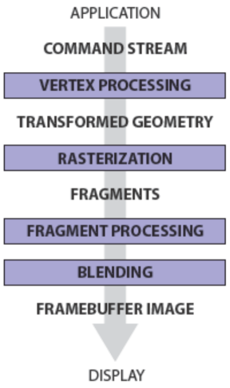


Figure 9.1. The stages of a graphics pipeline.

- Geometric objects are fed into the pipeline from an application, or scene description file, as a set of vertices.
- In the vertex processing stage, the vertices are operated on.
- After this, the vertices that represent the geometric primitives are sent off to be rasterized.
- The rasterizer breaks the primitive into fragments that correspond to the pixels covered by the primitive.
- The fragment processing stage processes the fragment before they are combined together in the fragment blending stage.

Section 9.1: Rasterization

- Terminology:
  - Rasterization - operation.
  - Rasterizer - performs the operation.
- Rasterizer performs two jobs for each primitive it receives:
  1. Iterates over the pixels that are covered by the primitive and marks them as covered by the primitive.
  2. Interpolates values (attributes) across the primitive.
- The outcome of rasterization is a set of fragments, one for each pixel covered by the primitive.

Section 9.1.1: Line Drawing

- Given two points: $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$ and $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$.
- The rasterizer should draw some “reasonable” pixels such that an approximate line is displayed.
  - Why approximate?
- Two types of line equations can be used:
  - Implicit - will be used for the remainder of the section.
  - Parametric
- Implicit line equation:

$$f(x, y) \equiv (y_o - y_1)x + (x_1 - x_0)y + x_o y_1 - x_1 y_0 = 0$$

- Assume that $x_0 < x_1$. If this is not the case, swap the points.
- The algorithm that we will use is the *midpoint* algorithm.

Midpoint algorithm

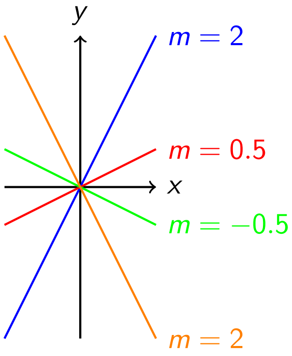
- The midpoint algorithm considers four distinct cases using the gradient m of the line:

Case 1: $m \in (0, 1]$

Case 2: $m \in (-\infty, -1]$

Case 3: $m \in (-1, 0]$

Case 4: $m \in (1, \infty)$



- What is special about each case?

- We will only look at $m \in (0, 1]$, as the others can be formed similarly.
- For this case, is it possible for an x-value to have more than one y-value?
  - What does this imply?

Algorithm 1 Midpoint algorithm for $m \in (0, 1]$

```
$y = y_0$
**for** $x = x_0$ to $x_1$ **do**
    draw(x,y)
    **if** some condition **then**
        $y = y + 1$
    **end if**
**end for**
```

- The *some condition* determines the "realisticness" of the line.
- The textbook suggests using the condition: $f(x+1, y+0.5) < 0$
  - This is the midpoint between the two possible pixels to the right of the current pixel.
    - $\begin{bmatrix} x+1 \\ y \end{bmatrix}$ or $\begin{bmatrix} x+1 \\ y+1 \end{bmatrix}$
- Note x and y are integers.
  - Why?
- Test type question:
  - Given some a line, and a condition, color the pixels that will be covered by the line as Fig 9.2.
- The remainder of the section discusses an efficiency improvement for the discussed line algorithm.

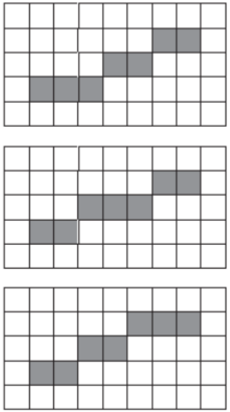


Figure 9.2. Three "reasonable" lines that go seven pixels horizontally and three pixels vertically.

Section 2.9: Barycentric

Given a triangle with points **a**, **b**, and **c**.

- Let **a** be the origin.
- Let the vector going from **a** to **b** (i.e. $\mathbf{b} - \mathbf{a}$) be the first axis and the vector going from **a** to **c** (i.e. $\mathbf{c} - \mathbf{a}$) be the second axis.
- i.e. we have a non-orthogonal set of basis vectors.
- This yields:

$$\mathbf{p}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

  With the constraint that:

$$\alpha + \beta + \gamma = 1$$

- Warning: Standard linear algebra does not work as expected with barycentric basis vectors
- See Section 2.9 of the textbook.

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}). \qquad (2.28)$$

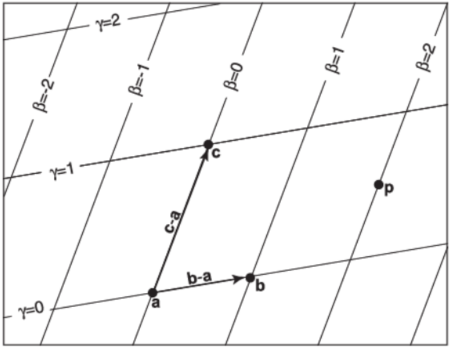


Figure 2.38. A 2D triangle with vertices **a**, **b**, **c** can be used to set up a nonorthogonal coordinate system with origin **a** and basis vectors (**b** – **a**) and (**c** – **a**). A point is then represented by an ordered pair (β, γ). For example, the point **p** = (2.0, 0.5), i.e., **p** = **a** + 2.0 (**b** – **a**) + 0.5 (**c** – **a**).

Section 9.1.2: Triangle Rasterization

- Triangles are defined by three points, and we would like to connect these points.
- We encountered similar problems with lines but these have a few twists.
- Coloring:
  - Say we would like to have one corner of the triangle blue, the next red and the last green.
  - How to interpolate the colors?

$$\mathbf{c} = \alpha \mathbf{c_0} + \beta \mathbf{c_1} + \gamma \mathbf{c_2}$$

  - Where:
    - c_0 to c_2 are colors.
    - α, β, γ are barycentric coordinates.
  - Known as *Gourand* interpolation.

- Another possible problem is that two adjacent triangles that share an edge, should not have a gap between them.
- This can be avoided by the following constraint:
  - Only draw a pixel **i.f.f.** the center of the pixel is inside the triangle.
  - i.e. the barycentric coordinates are between 0 and 1.
  - What if the centre of the pixel is perfectly on the edge?
    - Discussed later.
- Thus, the problem boils down to finding barycentric coordinates for the centre of pixels.

- The brute force algorithm is described by:

Algorithm 2 Bruteforce triangle rasterization algorithm

```
for all $x$ do
    for all $y$ do
        compute$(\alpha, \beta, \gamma)$ for $x, y$
        if $\alpha \in [0,1]$ and $\beta \in [0,1]$ and $\gamma \in [0,1]$ then
            $\mathbf{c} = \alpha \mathbf{c_0} + \beta \mathbf{c_1} + \gamma \mathbf{c_2}$
            drawPixel$(x, y, \mathbf{c})$
        end if
    end for
end for
```

- The textbook gives more optimised algorithm.

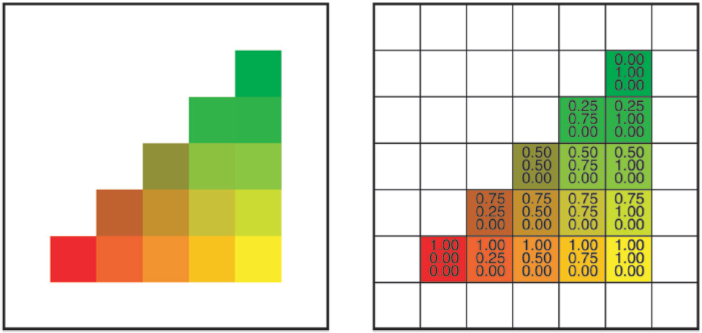


Figure 9.5. A colored triangle with barycentric interpolation. Note that the changes in color components are linear in each row and column as well as along each edge. In fact, it is constant along every line, such as the diagonals, as well.

- What about if the center of the pixel is on the edge between two triangles?
- Options?
  - Draw the pixel twice?
  - Don't draw the pixel?
  - Only draw the pixel once?
    - How?
    - Textbook suggests that one way is with an offscreen point.
    - Choose a point such that it will always be only on one side of the edge.
    - Whichever side the point is on gets to draw the pixel.
    - Still has one weakness, what is it?

- What about if the center of the pixel is on the edge between two triangles?
- Options?
  - Draw the pixel twice?
  - Don't draw the pixel?
  - Only draw the pixel once?
    - How?
    - Textbook suggests that one way is with an offscreen point.
    - Choose a point such that it will always be only on one side of the edge.
    - Whichever side the point is on gets to draw the pixel.
    - Still has one weakness, what is it?
    - Add a test for the case where the edge goes through the off-screen point.
- Textbook gives a more complex extension of the algorithm to account for the above discussion.
- Section 9.1.3 and Section 9.1.4 for now.

Section 9.2.1: Simple 2D Drawing

- In the simplest possible pipeline:
  - Nothing happens in the vertex or fragment stages.
  - In blending, the previous pixel color is just overwritten.
  - Application supplies primitives directly in pixel coordinates and the rasterizer does the rest.
  - Solid color is achieved by giving each vertex in the primitive the same color.

Section 9.2.2: A Minimal 3D Pipeline

- To modify our 2D pipeline to draw a 3D object, the following change needs to be made:
  - The vertex-processing stage does the following multiplication:
    - Incoming vertex position
    - Product of the modeling, camera, projection and viewpoint matrices.
- Painters algorithm:
  - Algorithm that determines the order that primitives need to be drawn, such that the objects at the back are behind the other objects, and the objects in front are in front of other objects.
  - Limitation: Intersections between objects and occlusion cycles.

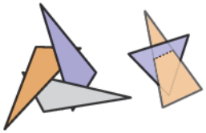


Figure 9.11. Two occlusion

Section 9.2.3: Using a z-Buffer for Hidden Surfaces

- Painter's algorithm is rarely used due to its inefficiency.
- Alternative: use z-buffer
  - At each pixel, keep track of the distance to the closest surface drawn.
  - If another fragment is further away then discard.
  - If another fragment is closer update the value.
- The z-buffer algorithm is implemented in the fragment blending phase.
- What is the initial value for each pixel at the start of the algorithm?
- The precision issues section are left to the curious student.

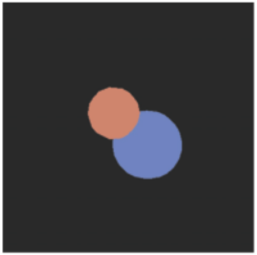


Figure 9.12. The result of drawing two spheres of identical size using the minimal pipeline. The sphere that appears smaller is farther away but is drawn last, so it incorrectly overwrites the nearer one.

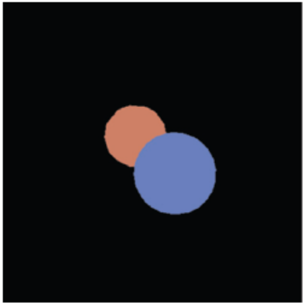


Figure 9.13. The result of drawing the same two spheres using the z-buffer.

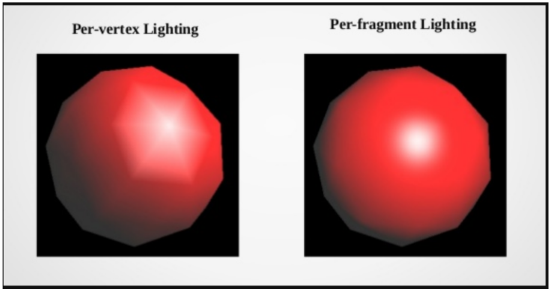
Section 9.2.4: Pre-vertex Shading

- Perform shading during the vertex stage.
- The application provides the:
  - Normals to the surfaces
  - Light position.
  - Light color.
- For each vertex, compute:
  - Viewer direction.
  - Light direction.
- The color is computed and then passed to the rasterizer as a vertex color.
- Called *Gouraud* shading.

Section 9.2.5: Pre-fragment Shading

- Perform shading during the fragment stage.
- Geometric information needed for shading, is passed through the rasterizer as attributes.
- Thus, coordination is needed between the vertex and fragment stages to prepare data.
- One approach:
  - Interpolate the eye-space surface normal and the eye-space vertex position.
  - Then it is used identically to pre-vertex shading
- The remainder of Section 9.2 will be covered later.

Comparison



https://osu-wams-blogs-uploads.s3.amazonaws.com/blogs.dir/4779/files/2021/10/5-1.png

Section 9.3: Simple Anti-aliasing

- The simple rasterization algorithms we discussed are also called standard or aliased rasterization.
- Basic idea is to allow pixels to be partly covered by primitives to create a blurry effect.
  - Helps the visual quality.
- Box filtering:
  - Take the average of all the colors assigned to a pixel.
  - Happens when a primitive partly covers the pixel.
- Super-sampling:
  - Create a higher resolution image then down-sample.

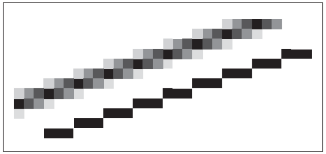


Figure 9.17. An antialiased and a jaggy line viewed at close range so individual pixels are visible.

- Section 9.4: Culling Primitives for Efficiency will be discussed later.
- The reason for this is due to viewing not being discussed yet.

Joke of the day - By ChatGPT

Why did the graphics pipeline break up with its girlfriend?

Joke of the day - By ChatGPT

Why did the graphics pipeline break up with its girlfriend?

Because every time they tried to render a relationship, it ended up pixelated and full of glitches!