Computer Graphics

Dr. rer. nat. Martin Möddel

April 6, 2021

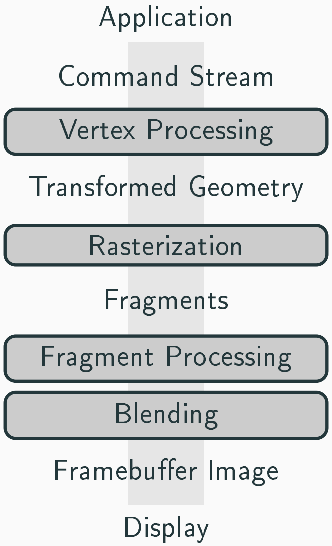
Institut für Biomedizinische Bildgebung

Object Ordered Rendering

Object Ordered Rendering

- In object ordered rendering each object is considered, one at a time
- Very efficient due to specialized hardware speeding up rendering
- Real time rendering possible
- Goals of graphic pipelines are different (performance, quality) and depend on the specific task
- Graphic APIs for real time rendering are e.g. Vulkan, OpenGL and Direct3D
- Graphic API for film production is e.g. RenderMan
- Despite different goals these pipelines share common fundamentals

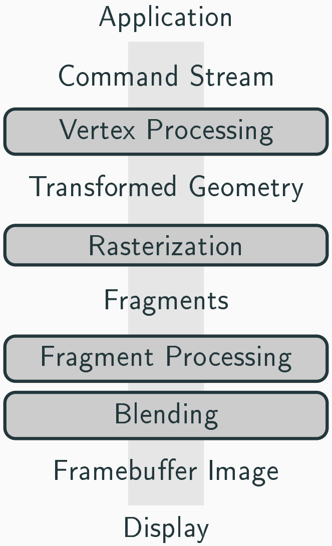
The stages of a graphics pipeline



Application

- Executed on main processor
- Handles user input
- Sends data and commands to graphics hardware
- Tasks such as
  - collision detection
  - animation
  - acceleration of pipeline
  - loading and removing of textures
- As technology advances more tasks are moved to specialized hardware

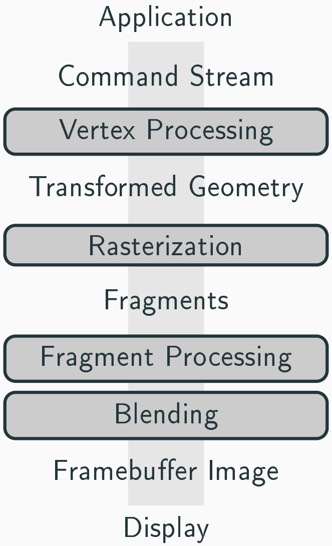
The stages of a graphics pipeline



Vertex Processor

- Processes vertices
- Creates new geometry from the vertices received and refines meshes at runtime
- Model and camera transformations
- Clipping
- Manipulates vertex lighting and color
- Viewport transformation to transform primitives to pixel space

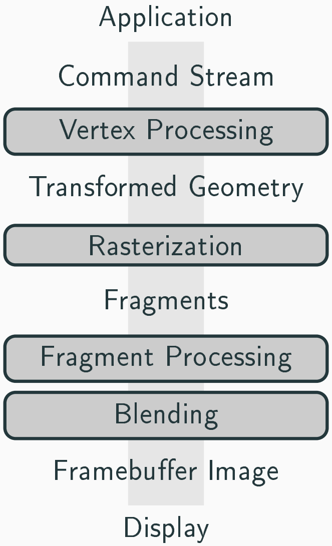
The stages of a graphics pipeline



Rasterization

- Processes each primitive
- Enumerates pixels covered by primitive and interpolates vertex attributes such as color and depth across primitive
- Outputs fragments, one for each pixel covered by the primitive
- Fragments are assigned to the corresponding pixels and hold certain attributes

The stages of a graphics pipeline



Blending

- Processes the fragments
- Colors each pixel according to the attributes of the fragments
- Saves the final image in framebuffer

Rasterization - Line Drawing

- Task is to colorize a set of pixels to approximate a line/linesegment
- line given by its two endpoints (x_0, y_0) and (x_1, y_1), where $x_0, y_0, x_1, y_1 \in \mathbb{R}$
- Basic algorithms use only one color, which often create aliasing artifacts
- Multiple shades of a color can be used to draw anti-aliased lines
- Convention for connectedness required
    - 8-connected (diagonal points count as connected)
    - 4-connected (only left, right bottom and top pixels count as connected)

Rasterization - Line Drawing

- Familiar form of a line is given by

$$y = mx + b \tag{1}$$

- Equivalently, it can be written in its implicit form

$$y - mx - b = 0 \tag{2}$$

- Or the more general form

$$Ax + By + C = 0 \tag{3}$$

  which can represent lines such as $x = 0$, where m would have to be infinite

Rasterization - Line Drawing

- Given the two points (x_0, y_0) and (x_1, y_1) A, B, and C must satisfy

$$Ax_0 + By_0 + C = 0 \tag{4}$$

$$Ax_1 + By_1 + C = 0 \tag{5}$$

- However, these two equations are not enough to obtain a unique solution for A, B, and C
- This ambiguity can be solved by setting the gradient of the implicit equation

$$(A, B) = (y_0 - y_1, x_1 - x_0) \tag{6}$$

  which is orthogonal to the vector $(x_1 - x_0, y_1 - y_0)$ pointing from (x_0, y_0) to (x_1, y_1)
- In this case $C = x_0 y_1 - x_1 y_0$

Rasterization - Line Drawing

- The implicit line equation is

$$f(x, y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0 \qquad (7)$$

- Assuming $x_0 < x_1$ (otherwise swap points) the slope $m = \frac{y_1 - y_0}{x_1 - x_0}$

Rasterization - Line Drawing

Direct Drawing

- Assume $m \in [-1, 1]$ (more run than rise) otherwise adapt algorithm
- Draw by direct evaluation of $y = m(x - x_0) + y_0$

```
$m = \frac{y_1 - y_0}{x_1 - x_0}$
**for** all integer $x$ in $[x_0, x_1]$ **do**
    $y = m(x - x_0) + y_0$
    round $y$ to nearest integer
    colorize pixel $(x, y)$
**end for**
```

- Lousy performance: y needs to be evaluated and rounded within each step
- How can other slopes $m \notin [-1, 1]$ be handled?

Rasterization - Line Drawing

Direct Drawing

- Increase performance by breaking up the main calculation

```
$y = y_0$
$m = \frac{y_1 - y_0}{x_1 - x_0}$
**for** all integer $x$ in $[x_0, x_1]$ **do**
    $y_I$ = round(y)
    colorize pixel $(x, y_I)$
    $y = y + m$
**end for**
```

- Avoids any multiplications inside hot loop

Rasterization - Line Drawing

Midpoint/Bresenham Algorithm

- Assume $m \in [0, 1]$ (more run than rise)
- Algorithm progresses from left to right (x increasing)
- For each pixel (x_i, y_i) drawn the next to draw to the right is either the same height $(x_i + 1, y_i)$ or one higher $(x_i + 1, y_i + 1)$
  - Check if line passes above or below midpoint $(x_i + 1, y_i + 0.5)$
  - If $f(x_i + 1, y_i + 0.5) < 0$ line passes above midpoint, else below

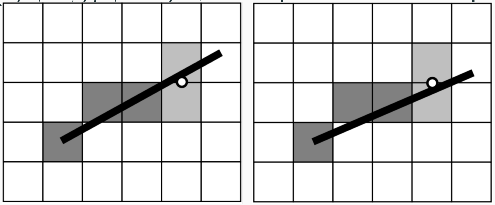


Figure 1: "Choosing between two possible pixels in Bresenham's line-drawing algorithm."

Rasterization - Line Drawing

Midpoint/Bresenham Algorithm

```
$y = y_0$
$d = f(x_0 + 1, y_0 + 0.5)$
**for** all integer $x$ in $[x_0, x_1]$ **do**
    colorize pixel $(x, y)$
    **if** d<0 **then**
        $y = y + 1$
        $d = d + (x_1 - x_0) + (y_0 - y_1)$
    **else**
        $d = d + (y_0 - y_1)$
    **end if**
**end for**
```

Rasterization - Line Drawing

Symmetric Extension of Algorithms

- Most algorithms pose restrictions uppon the slope of the line to be drawn
- Often symmetry can be used to extend algorithm to all slopes

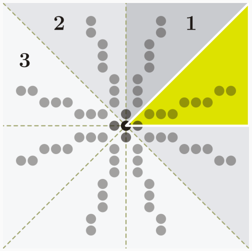


Figure 2: "A scan-converted line and the use of symmetry to draw similar lines in the other octants." by Phrood licensed under CC BY-SA 3.0

Rasterization - Triangles

Triangle Rasterization

- A typical entity to rasterize is the triangle
- More general polygons can be broken down into triangles and are therefore also covered
- When rendering adjecent triangles algorithm should leave no holes between the triangles
- Pixels on the Edge of adjecent triangles should be rasterized no more than once to such that the result does not depend on the order in which the triangles are rasterized

Rasterization - Triangles

Barycentric Coordinate System

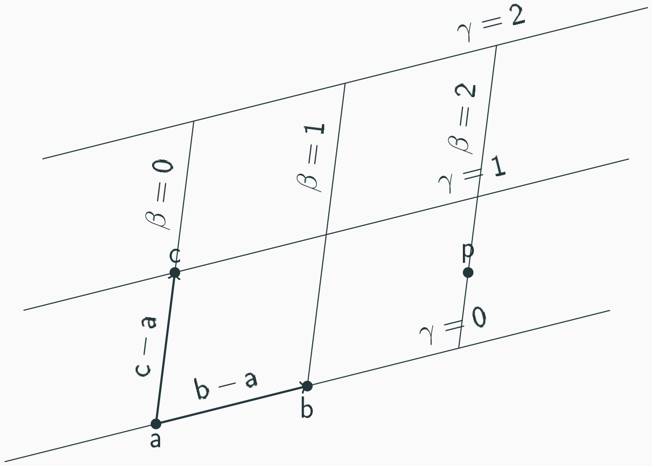
- Consider a triangle with its three vertices a, b, and c
- In general the two sides c − a and b − a span a non-orthogonal basis
- Each point p can be written as

$$\begin{aligned} p &= a + \beta(b - a) + \gamma(c - a) && (8) \\ \Leftrightarrow p &= \underbrace{(1 - \beta - \gamma)}_{\alpha} a + \beta b + \gamma c, && (9) \end{aligned}$$

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

Rasterization - Triangles

Barycentric Coordinate System



Rasterization - Triangles

Barycentric Coordinate System

- Even if only two parameters are independent all three are usually used, e.g. for the interpolation of vertex colors across the triangle
- Points inside the triangle satisfy

$$0 < \alpha < 1 \tag{10}$$
$$0 < \beta < 1 \tag{11}$$
$$0 < \gamma < 1 \tag{12}$$

- Points lie on an edge if one parameter (α, β or γ) is 0 and the others are in $(0, 1)$
- Points lie on a vertex if two parameters are 0 and one is 1

Rasterization - Triangles

Computation of Barycentric Coordinates

- Given a point p its barycentric coordinates (α, β, γ) can be computed by solving

$$\begin{pmatrix} b_x - a_x & c_x - a_x \\ b_y - a_y & c_y - a_y \end{pmatrix} \begin{pmatrix} \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} p_x - a_x \\ p_y - a_y \end{pmatrix} \tag{13}$$

  and setting $\alpha = 1 - \beta - \gamma$
- Closed formulas exist for the solution of the linear system

Rasterization - Triangles

Gouraud Shading

- The inner of a 2D triangle with vertices a, b, and c can be drawn by calculating the barycentric coordinates (α, β, γ) of all screen coordinates and colorizing all points, which lie inside the triangle
- If we have colors c_a, c_b and c_c assigned to the vertices, we can linearly interpolate these colors inside the triangle by

$$c = \alpha c_a + \beta c_b + \gamma c_c \tag{14}$$

- This interpolation method is referred to as Gouraud shading
- This algorithm can be enhanced by restricting the main loop over the screen pixels to a 2D bounding box containing the triangle

Rasterization - Triangles

Gouraud Shading

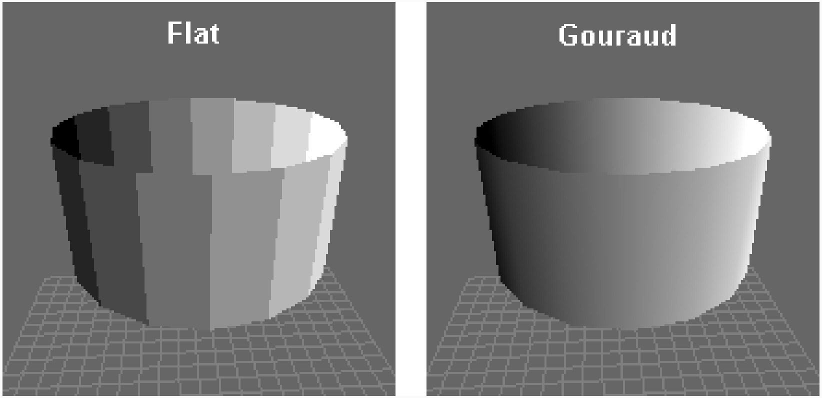


Figure 3: "Comparison of flat shading and Gouraud shading."

Rasterization - Triangles

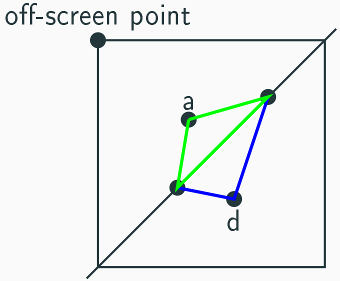
Edge Drawing

- With Gouraud shading only the triangle inside can be drawn
- This might create holes in objects consisting of multiple triangles, which share edges
- Drawing the edges for all triangles introduces a dependency, where the results depend on the drawing order of the triangles
- Remove ambiguity and draw edge only once
- There are many strategies to do so

Rasterization - Triangles

Edge Drawing - Robust Strategy

- Choose off-screen point
- Unless the line through common vertices runs through this point only one triangle will be on the off-screen point side
- Draw the edge for this triangle only
- Handle the exeption using a secondary off-screen point



Rasterization

Remarks

- In principle each shape can be rasterized
- Triangles and lines are most prominent because of their simplicity/performance
- Specialized hardware excels at rasterizing triangles
- To make use of the hardware scenes and objects have to be approximated by triangles (an interesting mathematical problem, which will be discussed later)