The Computer Graphics Pipeline

- Programmable vs. Fixed Function Pipeline
- SIMD
Chapter 3

1. Introduction
2. The Computer Graphics Pipeline
3. **Object Representation**
4. Object Transformation
5. 3D – Projections, Camera, and Lighting
6. Scene Representation and Interaction
7. Advanced Texturing and Shading
Input - From Objects to Images

• How do we represent objects on the computer?

• Fulfill a number of requirements
  • Low memory consumption
  • Easy application of transformation operations (translate, clip,…)
  • Closed representation (Object + Operation = Object)
  • Easy to render

• Answer: Compose object of simpler base elements
  • Vertices, Lines, Polygons (Triangles, Quads, …)
  • Object represented as mesh/tessellation
  • Additional optical properties (color)
Input - Basic Elements

- Vertex, Line, Triangle

Drawing in OpenGL (fixed function - not using VBO):

```gl
glBegin(GL_POINTS);
glVertex2f(x,y);
glEnd();

glBegin(GL_LINES);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();

glBegin(GL_TRIANGLES);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glEnd();
```
Input - Basic Elements

Chrschn – Wikimedia Commons
This chapter assumes a static 2D world
  - After transformations and clipping

What does our input look like?
How do we show it on screen?
Rasterization

- Rasterization of 2D lines and triangles

- At this stage vertices are given in **screen coordinates** (2D)
- Attributes (color, depth, etc.) given per vertex
Rasterization

- Create fragments (potential pixels)
  - Interpolate attributes (color, depth, etc.) from vertices
  - Identify target pixels
    - Integer grid – centered at 0 or 0.5 (OpenGL)
    - Here: pixels are squares
Rasterization

- **DDA algorithm** (digital differential analyzer) for lines

\[
\frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = m \quad 0 \leq m \leq 1
\]

\[
\Delta y = m \Delta x
\]

float y = y_1;
for(int x = x_1; x <= x_2; ++x, y+=m)
setPixel(x,round(y));
Rasterization

• DDA algorithm
  • Find best y for each x
  • Slope condition is important (cover symmetric cases)

\[
0 \leq m \leq 1
\]

```plaintext
float y = y_1;
for(int x = x_1;x <= x_2;++x, y+=m)
  setPixel(x,round(y));
```
Rasterization

- **Bresenham’s algorithm** avoids floating point operations

\[ y = \frac{\Delta y}{\Delta x} x + b \quad \quad f(x, y) = x\Delta y + b\Delta x - y\Delta x = 0 \]

\[ 0 \leq m \leq 1 \quad \text{decide between } y_i \text{ and } y_i+1 \]

Decision based on sign of \[ f(x + 1, y + 0.5) \]
Rasterization

\[ f(x, y) = x\Delta y + b\Delta x - y\Delta x = 0 \]

\[ D_0 = f(x_0 + 1, y_0 + 0.5) - f(x_0, y_0) \]

incremental decision variable

\[ D_0 = \Delta y - \frac{1}{2}\Delta x \]

if \( D \geq 0 \) pick upper candidate

\[ D_{k+1} = D_k + \begin{cases} 
\Delta y & \text{if } D_k < 0 \\
\Delta y - \Delta x & \text{else}
\end{cases} \]

multiply \( D \) by 2 to ensure integers
Rasterization

• Some examples
Rasterization

- Anti-aliasing (AA)
- Super sampling AA (SSAA)
- Multisample AA (MSAA)
- …
Rasterization

- Rasterization of polygons

- Fill inside of polygon
- What is “inside”?  
- Orientation: Clockwise, Counterclockwise
Rasterization

• What fragment positions are “inside”? 

odd-even test

winding number
Rasterization

- Scanline algorithm

- Flood fill, ...

- Rasterizer interpolates attributes from vertices to fragments
What about colors?

- Colors (and other attributes) are specified at vertices
- How do we find values inside of polygons?
Interpolation

- **Interpolation** constructs “new” data within the range of known data points
- Construct a function with continuous domain from discrete data
Interpolation

• Mathematical formulation:

Given \((x_i, f_i)\) ("positions" and "values")

Find function \(f\) defined on \([x_0, x_n]\)

such that \(f(x_i) = f_i\)
Interpolation

- We have already seen that there can be multiple solutions
- How do we pick the “right” function?

Answer: Introduce constraints and/or make use of knowledge about data
Interpolation: Side Note

• Related concepts:

Approximation

Extrapolation
Interpolation: Linear

- Linear interpolation

Interpolation conditions

\[ f(x_0 = 0) = f_0 \]
\[ f(x_1 = 1) = f_1 \]

Linear interpolation

\[ f(x) = a + mx \]

Solution

\[ f(x) = f_0 + (f_1 - f_0)x \]

Alternative form

\[ f(u) = (1 - u)f_0 + uf_1 \]
Interpolation: Linear

- Linear interpolation

Alternative form

\[ f(u) = (1 - u)f_0 + uf_1 \]

Values between \( x_0 \) and \( x_1 \) are “mixtures” of \( f_0 \) and \( f_1 \)

\[ u = 0.2 \quad \rightarrow \quad 80\% f_0 + 20\% f_1 \]
Interpolation: Linear

- Basis functions/Blending functions

\[ f(u) = (1 - u)f_0 + uf_1 \]

\[ (1 - u) + u = 1 \]  \text{partition of unity}

\[ f(u) = b_0(u) \cdot f_0 + b_1(u) \cdot f_1 \]

\[ f(u) = \sum_{i} b_i(u) \cdot f_i \]  \text{control point contributions are blended together}
What about colors?

- Linear interpolation (component-wise)

\[ f_{RGB}(t) = (f^R(t), f^G(t), f^B(t)) \]