Rendering Pipeline
VFX VFX VFX VFX VFX!!!

- Turkish Airlines
- Zach King
Rendering

- Converting a 3D scene to a 2D image
Rendering

- Converting a 3D scene to a 2D image
- Basic rendering tasks:
  - **Modeling**: creating the “world” or “scene”
  - **Viewing**: determining how the “camera” or “eye” sees
  - **Visibility**: determining what is visible and what is occluded
  - **Shading**: determining the color of each visible 3D element (and 2D pixel)
Rendering Scenarios

- Non-interactive batch rendering
- Interactive online rendering
Non-Interactive Rendering

- Used in motion picture industry, visualizations
- High level of details for a specific set of parameters
- Quality is highest priority
- Typically uses ray tracing
Image Characteristics

- Reflections
- Refractions
- Hard & Soft Shadows

Hard to Achieve:
- Motion blur
- Depth of Field
Time Consuming

- Typical Scene at
  - 100s of lights
  - 1,000s of textures – too many to fit in memory!
  - 10,000s of objects
  - 100,000,000s of polygons – too many to fit!
  - Shaders with 10,000s lines of code
Interactive Online Rendering

- Games industry, interactive visualization
- High speed rendering for changing parameters (e.g. view)
- Speed (interactivity) is highest priority (best quality for a given time budget)
- Low cost hardware support (started for games but used for various purposes)
Hardware Support

- Most computers today include a “video card”.
- Today it is called a GPU – graphics processing unit
- It is in fact a collection of hardware implementations of a rendering pipeline. (some GPU support for ray tracing is available)
The 3D Rendering Pipeline

This is a pipelined sequence of operations to convert a set of 3D primitives into a 2D image.
The 3D Rendering Pipeline

- **3D Geometric Primitives**
- **Modeling Transformation**
  - Transform into 3D world coordinate system
- **Lighting**
  - Illuminate according to lighting and reflectance
- **Viewing Transformation**
  - Transform into 3D camera coordinate system
- **Projection Transformation**
  - Transform into 2D camera coordinate system
- **Clipping**
  - Clip primitives outside camera’s view
- **Scan Conversion**
  - Draw pixels (includes texturing, hidden surface, etc.)
Basic Display Primitives

- The rendering pipeline supports two basic display primitives:
  1. Pixels (e.g. from images) are used directly or combined into the image.
  2. Triangles are converted to pixels using vertex information by rasterization and interpolation.
Why Triangles?

- Most simple polygon, a planar polygon.
- Simplifies hardware design and implementation.
- Every polygon can be triangulated.
- Higher degree surfaces can be approximated to any extent by triangles.
- Bilinear interpolation gives reasonable results (color, position, normal).
Special Hardware Support for Rendering Triangles!

Raster Screen

3D Projection

2D Rasterization

Virtual World Models
From Objects to Pixels

Every geometric primitives is converted to triangles and then to pixels

Model → Triangulation → Rasterization → Pixels
Key Idea of the Rendering Pipeline

- Objects are converted to polygons (triangles)
- Instead of dealing with triangles we deal only with vertices of the triangles in 3D (transformation & projection) until we project to 2D
- In 2D we fill in the triangles (rasterization)
Rendering Triangles

- Step 1: Project triangles to screen
- Step 2: Rasterize triangles to pixels
Transformations!

3D Geometric Primitives

- **Modeling Transformation**: Transform into 3D world coordinate system
- **Lighting**: Illuminate according to lighting and reflectance
- **Viewing Transformation**: Transform into 3D camera coordinate system
- **Projection Transformation**: Transform into 2D camera coordinate system
- **Clipping**: Clip primitives outside camera’s view
- **Scan Conversion**: Draw pixels (includes texturing, hidden surface, etc.)

(Image)
Transforming a Point

Transformations map points from one coordinate system to another.

- **Modeling Transformation**
  - 3D Object Coordinates
- **Viewing Transformation**
  - 3D World Coordinates
- **Projection Transformation**
  - 3D Camera Coordinates
- **Window-to-Viewport Transformation**
  - 2D Screen Coordinates
  - 2D Image Coordinates
  - p'(x', y')
Coordinate systems

- **Object space**
  - local to each object

- **World space**
  - common to all objects

- **Eye space / Camera space**
  - derived from view frustum

- **Screen space**
  - indexed according to hardware attributes
Transformations Lectures

- 2D transformations
- 3D transformations
- Projections
Visibility & Shading

- Modeling and Viewing are covered by transformations
- Two more rendering tasks:
  - Visibility determination: what is visible and what is occluded.
  - Shading: what is the color of each visible 3D element (and 2D pixel)
Visibility in GPU Rendering

- Project all triangles & convert to pixels
- But… must remember the pixel “depth” = distance from camera
- Can we do it in any order?
Z-Buffer Algorithm

- Any pixel rendered holds a color and a distance from the viewer (z).
- Pixels are accumulated in a buffer using the z-buffer algorithm:

  Clear Z buffer (set z to infinity and store bg-color)
  For each triangle in scene
    Convert triangle to pixels
    For each new pixel P(i,j) rendered
      If the distance $z_P$ of P is smaller than $Z(i,j)$
        store the $z_P$ & color of P in $Z(i,j)$
  Send the buffer to display
Z-Buffer

A simple three dimensional scene

Z-buffer representation
Frame Buffer

Frame Buffer (3 for color)
Frame Buffer Refresh

Refresh rate is usually 30-75Hz
Double Buffering

- Problem: drawing takes time. You don’t want to show “half drawn pictures or objects”
- When things move fast it can create flickering effect
- Solution: use two buffers
  - Draw your world into one frame buffer memory while the second is displayed (used for refresh)
  - When you finish drawing you switch between the two and draw into the second
  - Continue the same way by switching after each drawing
Rasterization: from Vertexes to Pixels

- Triangles are planar, this means a simple bilinear interpolation can be used to interpolate the attributes from the vertices to any pixel inside.
Interpolation?

- Determining values inside the triangle based on its vertices only!

- How?

Convex combination ➔ Barycentric Coordinates:

\[ \alpha_1 + \alpha_2 + \alpha_3 = 1 \]
\[ p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 \]
\[ F(p) = \alpha_1 F_1 + \alpha_2 F_2 + \alpha_3 F_3 \]
Barycentric Coordinates?

- For triangles (and tetrahedra in 3D etc.) the convex combination defines barycentric coordinates for each point inside the triangle:

\[ p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 \]

\[ \alpha_1 = \frac{A_1}{A_0} \]
\[ \alpha_2 = \frac{A_2}{A_0} \]
\[ \alpha_3 = \frac{A_3}{A_0} \]
\[ \alpha_1 + \alpha_2 + \alpha_3 = 1 \]
Barycentric Coordinates

\[
x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \\
y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3
\]

\[
x = \lambda_1 x_1 + \lambda_2 x_2 + (1 - \lambda_1 - \lambda_2)x_3 \\
y = \lambda_1 y_1 + \lambda_2 y_2 + (1 - \lambda_1 - \lambda_2)y_3
\]

\[
\lambda_1(x_1 - x_3) + \lambda_2(x_2 - x_3) + x_3 - x = 0 \\
\lambda_1(y_1 - y_3) + \lambda_2(y_2 - y_3) + y_3 - y = 0
\]

\[
T = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix} \quad T \cdot \lambda = r - r_3 \quad \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = T^{-1}(r - r_3)
\]

\[
\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{\det(T)} = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)},
\]

\[
\lambda_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{\det(T)} = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_3 - y_1)(x_2 - x_3) + (x_1 - x_3)(y_2 - y_3)},
\]

\[
\lambda_3 = 1 - \lambda_1 - \lambda_2.
\]
Simpler: 
(Bi-)Linear Interpolation

- Assumption: Data varies linearly between adjacent data points (vertices).
- On triangle edges:
  \[(1-t)f_0 + tf_1\] when \(0 \leq t \leq 1\)
- Linear interpolation (a mapping from \([0,1]\)):
- What happens inside the triangles?
- Interpolate linearly between the edge points!
- Done in 2D!
Other Vertex Information

- Position in space (Depth)
- Normal
- Color/Shading
- Texture map coordinate
Summary

- **Triangulation**: Object to triangles
- **Transformation & projection**: just 3D vertices to 2D vertices
- **Illumination**: calculate the color of the vertex (pixel)
  \[ I_\lambda = I_{a\lambda}k_aO_{d\lambda} + \sum_i f_{o_{ii}}I_{p_{ii}\lambda}[k_dO_{d\lambda}(\vec{N} \cdot \vec{L}_i) + k_sO_{s\lambda}(\vec{R}_i \cdot \vec{V})^n] \]
- **Visibility**: z-buffer
- **Rasterization**: calculate the position and color of all pixels from the vertices using bilinear interpolation