Overview: Interactive Computer Graphics

- Interactive Graphics System Model
- Graphics Pipeline
- Coordinate Systems
- Modeling Transforms
- Cameras and Viewing Transform
- Lighting and Shading
- Color
- Rendering
- Visible Surface Algorithms
- Rasterization
- Hardware
The Viewing Pipeline

- **3D Model**
  - Model coords

- **3D World**
  - World Coords

- **Viewing Transform**
  - Camera coords
  - Camera Specification

- **3D Clipping**
  - Clipping coords

- **Rendering**
  - Display
  - Lighting Specification

- **Projection to 2D**
  - Screen coords
The Viewing Pipeline

Figure 3.1: The graphics pipeline for display traversal.
Coordinate Systems

- Model Coordinates
- World Coordinates
- Viewing (Camera, Eye) Coordinates
- Perspective/Image Coordinates
- Screen/Display Coordinates
Graphics Primitives/Attributes

Primitives

- Point, Line, Polyline, Polygon, Rectangle, Circle
- Cubic Curves and Surfaces (Bezier, B-Spline, NURBS)
- Marker, Polymarker, Text
- Texture (2D, 3D)

Attributes

- Lines/Markers: style, width
- Color (R,G,B or index into lookup table)
- Filled Primitives: style, pattern
- Text: style (font), size, origin
Affine Transforms

- Most common in computer graphics.
- Permits a simple matrix representation
- Can be composed

Definition

\[
\begin{bmatrix}
Q_x & Q_y & Q_z \\
\end{bmatrix} =
\begin{bmatrix}
a & d & g \\
b & e & h \\
c & f & k \\
\end{bmatrix}
\begin{bmatrix}
P_x \\
P_y \\
P_z \\
\end{bmatrix} +
\begin{bmatrix}
T_x \\
T_y \\
T_z \\
\end{bmatrix}
\]

\[
\vec{Q} = \vec{M} \vec{P} + \vec{T}
\]

Primitive Affine Transforms: Scale, Rotate, Translate
Homogeneous Coordinates

- Allows all 3 primitive transforms to use a matrix representation.
- Add a fourth coordinate, \( w \); additional column and row to matrix.
- To convert \( P_H \) to \( P_{2D} \), divide each coordinate by the \( w \) and discard the 3rd coordinate.
- \( w = 0 \) represents points at infinity.

\[
\begin{align*}
P_H & = (P_x, P_y, P_z, P_w) \\
P_{3d} & = (P_x/P_w, P_y/P_w, P_z/P_w)
\end{align*}
\]
Modeling Transformations

Translation

\[ T(t_x, t_y, t_z) = \begin{bmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & t_z \\
0 & 0 & 0 & 1
\end{bmatrix} \]

Scale

\[ S(s_x, s_y, s_z) = \begin{bmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & s_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \]
Modeling Transformations

Rotation

\[ R_z(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \]

\[ R_x(\theta) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \]

\[ R_y(\theta) = \begin{bmatrix}
\cos \theta & 0 & \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \]
Cameras and 3D Viewing

- A mechanism for specifying the viewer location, orientation and volume of the world that is of interest.
- A pin-hole camera model is assumed.
Camera Parameters

- View Reference Point, $V\vec{R}P$.
- Look At vector, $\vec{N}$.
- Up Vector, $\vec{V}$.
- Vector $\vec{U} = \vec{N} \times \vec{V}$
- Window
- Camera (viewer) position.
- Near and Far clipping planes.

These parameters specify a view volume (frustum) that is a truncated pyramid.
View Volume

- The volume of 3D space that is projected and displayed.
- Volume defined by a 2D window on the projection plane and **front** and **back** clipping planes (hither/yon, near/far).
- View volume is specified in camera coordinates.
- View volume is a **truncated frustum**.
Viewing Transform

- Object points in world coordinates are transformed into the camera coordinate system with
  - The viewer at the origin.
  - Look-at vector along the negative $Z$ axis.
  - The Up vector along the $Y$ axis.
Viewing Transform

The viewing transform is an affine transform, consisting of a rotation and translation, as follows:

$$ M_{wv} = \begin{bmatrix} u_x & u_y & u_z & r'_x \\ v_x & v_y & v_z & r'_y \\ n_x & n_y & n_z & r'_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

with $\vec{r}' = (-\vec{r}.\vec{U}, -\vec{r}.\vec{V}, -\vec{r}.\vec{N})$
Geometric Projections

Projection

“An operation that transforms points in N-dimensional space to another of dimensionality less than n”

Geometric Projection

Application to geometric coordinate systems (Cartesian, Polar, Spherical and Cylindric systems)

Types of Projections

■ Planar
■ Non-Planar

Application to Computer Graphics

1. Viewing 3D environments on a 2D screen.

2. Visualization of multi-variate data from scientific applications.
Planar Projections
Planar Projections

Parallel Projection
- Projectors are parallel to each other.

Perspective
- Projectors converge to a point.
Perspective Projection

- Projectors converge to a point.
- All points on the same projector project to the same point, destroying depth information.
- Depth information must be carried along the pipeline for use in visible surface calculations.
- Perspective transformation effectively involves a division of the 3D point coordinates by its depth (from the view point).
Perspective Transformation

If \( \vec{P} = (p_u, p_v, p_n) \) is an object point, \( \vec{E} = (e_u, e_v, e_n) \) is the camera position and \( \vec{P}' = (p'_u, p'_v, p'_n) \) is the projection point, assuming the projection plane contains the origin of the camera coordinate system, it can be shown that

\[
\begin{align*}
    p'_u &= \frac{p_u}{1 - p_n/e_n} \\
    p'_v &= \frac{p_v}{1 - p_n/e_n}
\end{align*}
\]
Assuming \( p'_n = \frac{p_n}{1-p_n/e_n} \) as a pseudo-depth, we can write

\[
P' = (p_u, p_v, p_n, 1 - p_n/e_n)
\]

\[
= [p_u, p_v, p_n, 1] \vec{M}_p
\]

with

\[
\vec{M}_p = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & -1/e_n \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Lighting and Shading

The Rendering Equation

\[ I(x, x') = g(x, x') \left[ \epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right] \]

where

- \( I(x, x') \) = Intensity of light passing from \( x' \) to \( x \).
- \( g(x, x') \) = “Geometry term”
- \( \epsilon(x, x') \) = Emitted light intensity from \( x' \) to \( x \).
- \( \rho(x, x', x'') \) = Light intensity scattered from \( x'' \) to \( x \) by a patch of surface at \( x' \).
Local Lighting Model

- Consists of 3 components, ambient, diffuse and specular

**Ambient Reflection**

\[ I_a = k_a I_a \]

- \( I_a \) is the ambient intensity, constant for all objects, and \( k_a \), the ambient reflection coefficient.
- Use to model illumination that arrive at object points through indirect sources, such as nearby objects through multiple reflections or scattering.
Diffuse Reflection

\[ I_d = I_p k_d \cos \theta \]
\[ = I_p k_d (\vec{N} \cdot \vec{L}) \]

where \( \vec{N} \) is the unit normal to the surface and \( \vec{L} \), the vector (of unit length) to the light source, \( I_p \) the light source intensity.

- We approximate diffuse reflection using Lambert’s law.
- Intensity of light reflected is a constant, and independent of viewer position.
Specular Reflection (Phong Model)

\[ I_s = k_s \cos^n \alpha \]
\[ = k_s (\vec{R} \cdot \vec{V})^n \]
Local Illumination Model

Including ambient and diffuse reflections, the local lighting model becomes

\[ I = I_a + I_d + I_s \]
\[ I = k_a I_a + I_p [k_d (\vec{N} \cdot \vec{L}) + k_s (\vec{R} \cdot \vec{V})^n] \]

Assuming a wavelength based distribution of light and \( l \) light sources,

\[ I_\lambda = I_{a\lambda} k_a + \sum_{1}^{l} f_{att_i} I_{p\lambda_i} [k_{d\lambda} \cos(\vec{N} \cdot \vec{L}_i) + k_{s\lambda} (\vec{R}_i \cdot \vec{V})^n] \]
Rendering Algorithms: Gouraud Shading

- Face Normals are averaged to obtain vertex normals.
- Vertex Intensities \((I_1, I_2, I_3)\) are obtained by application of a lighting model.
Rendering Algorithms: Phong Shading

- Compute Normals at vertices of polygon.
- Interpolate each component of normal at any interior point of polygon.
- Apply illumination model.
- Can easily be incorporated into a scanline algorithm.
Visible Surface Algorithms

Z (Depth) Buffer Algorithm (Catmull ’74)

- A buffer of Z or depth values is maintained, one for each pixel.
- During scan conversion, if primitive’s depth is less (closer) than the existing value in the Z buffer, replace it and corresponding color (intensity) in the frame buffer.
- Each primitive is scanconverted once, independent of any other primitive and in any order.
- Z (depth) can be calculated from the polygon plane equation, or interpolated from depth at polygon vertices
Z-Buffer Algorithm

For each object primitive
{
    For each pixel in projection
    {
        Z = primitive’s depth value at pixel(x,y).
        if Z < Z_BUF[x][y]
        {
            Write_ZBUF (x, y, Z)
            Write_Pixel (x, y, pixel_color)
        }
    }
}
Rasterization

Scanline Polygon Fill Algorithm

With some modifications and additional parameters, is used to scanconvert polygons along with the Z buffer algorithm

Approach

■ Exploit the geometry of the polygon.
■ Use scanline and edge coherence for efficient filling.
Color Models

RGB Color Model

Application

- Color CRT Monitors.
CMY Color Model

- Complement of the RGB model.
- Uses *subtractive* primaries, Cyan, Magenta, Yellow.

\[
\begin{bmatrix}
C \\
M \\
Y
\end{bmatrix} = \begin{bmatrix} 1 \\
1 \\
1
\end{bmatrix} - \begin{bmatrix} R \\
G \\
B
\end{bmatrix}
\]

Application

- HardCopy Devices, such as ink jet color printers.
HSV Color Model

- Model defined in a cylindrical coordinate system.
- Colors are defined in a hexcone or six-sided pyramid.
- Based on the artist’s model of tints, shades and tones.

⇒ **Hue** is measured around the vertical axis.
⇒ **Saturation** ranges from 0 on the centerline to 1 on the triangular sides.
⇒ **Value** ranges from 0.0 (black) at the apex of the pyramid to 1.0 (white) at the base.
Video Controller reads the frame buffer through a special port.

A single address space system.

Memory contention is still a problem, but the flexibility of a single address space (for raster ops, for instance) outweighs this factor.
Graphics Processing Units (GPU)

- Modern graphics cards (Nvidia GEForce, ATI Radeon etc) are extremely powerful computing platforms for processing/shading large amounts of geometry
- Pipelined architecture
- Support for 3D textures (multitexturing, dependent texturing)
- May be used for general purpose computing
- Examples: Accelerated ray tracing, solving linear systems, fluid flow.