Graphics Pipeline: Transformation, Shading/Lighting, Projection, Texturing, and more!

Fall 2023

Daniel G. Aliaga
Department of Computer Science
Purdue University
Computer Graphics Pipeline

Geometry

- Modeling Transformation: Transform into 3D world coordinate system
- Lighting: Simulate illumination and reflectance
- Viewing Transformation: Transform into 3D camera coordinate system
- Clipping: Clip primitives outside camera’s view
- Projection: Transform into 2D camera coordinate system
- Scan Conversion: Draw pixels (incl. texturing, hidden surface…)

Image
Computer Graphics Pipeline

Geometry

Modeling Transformation
- Transform into 3D *world* coordinate system

Lighting
- Simulate illumination and reflectance

Viewing Transformation
- Transform into 3D *camera* coordinate system
- Clip primitives outside camera’s view

Clipping
- Transform into 2D camera coordinate system

Projection

Scan Conversion
- Draw pixels (incl. texturing, hidden surface…)

Image
Modeling Transformations

• Most popular transformations in graphics
  – Translation
  – Rotation
  – Scale
  – Projection

• In order to use a single matrix for all, we use homogeneous coordinates...
Modeling Transformations

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

Identity

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
sx & 0 & 0 & 0 \\
0 & sy & 0 & 0 \\
0 & 0 & sz & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

Scale

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & tx \\
0 & 1 & 0 & ty \\
0 & 0 & 1 & tz \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

Translation

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
-1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

Mirror over X axis
Modeling Transformations

Rotate around Z axis:

\[
\begin{bmatrix}
x'
\end{bmatrix} =
\begin{bmatrix}
\cos \Theta & -\sin \Theta & 0 & 0 \\
\sin \Theta & \cos \Theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x
\end{bmatrix}
\]

Rotate around Y axis:

\[
\begin{bmatrix}
x'
\end{bmatrix} =
\begin{bmatrix}
\cos \Theta & 0 & -\sin \Theta & 0 \\
0 & 1 & 0 & 0 \\
\sin \Theta & 0 & \cos \Theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x
\end{bmatrix}
\]

Rotate around X axis:

\[
\begin{bmatrix}
x'
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \Theta & -\sin \Theta & 0 \\
0 & \sin \Theta & \cos \Theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x
\end{bmatrix}
\]

And many more…
Computer Graphics Pipeline

Geometry

- Modeling Transformation: Transform into 3D world coordinate system
- Lighting: Simulate illumination and reflectance
- Viewing Transformation: Transform into 3D camera coordinate system
- Clipping: Clip primitives outside camera’s view
- Projection: Transform into 2D camera coordinate system
- Scan Conversion: Draw pixels (incl. texturing, hidden surface...)

Image
Diffuse

(mostly)
Specular++
Environment Mapping
Subsurface Scattering

(a) High-res geometry  (b) Real-time hybrid map rendering  (c) Offline SSS rendering
Others

Transparency

Radiosity

Ambient occlusion
Others
Lighting and Shading

• Light sources
  – Point light
    • Models an omnidirectional light source (e.g., a bulb)
  – Directional light
    • Models an omnidirectional light source at infinity
  – Spot light
    • Models a point light with direction

• Light model
  – Ambient light
  – Diffuse reflection
  – Specular reflection
Lighting and Shading

• Diffuse reflection
  – Lambertian model
Lighting and Shading

• Diffuse reflection
  – Lambertian model
Lighting and Shading

- Diffuse reflection
  - Lambertian model

$$I_D = K_D (N \cdot L) I_L$$
Lighting and Shading

- Specular reflection
  - Phong model
Lighting and Shading

• Specular reflection
  – Phong model

\[ I_S = K_S (V \cdot R)^n I_L \]
Lighting and Shading

• Specular reflection
  – Phong model
Computer Graphics Pipeline

Geometry

- **Modeling Transformation**: Transform into 3D *world* coordinate system
- **Lighting**: Simulate illumination and reflectance
- **Viewing Transformation**: Transform into 3D *camera* coordinate system
- **Clipping**: Clip primitives outside camera’s view
- **Projection**: Transform into 2D camera coordinate system
- **Scan Conversion**: Draw pixels (incl. texturing, hidden surface…)

Image
Viewing Transformation

\[ \begin{align*}
\tilde{x}_c &= R(\tilde{X} - C) \\
\tilde{x}_c &= R\tilde{X} - RC
\end{align*} \]

World-to-camera matrix \( M \)

\[ R = R_x R_y R_z \]
3x3 rotation matrices

\[ t = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^T \]
translation vector
Computer Graphics Pipeline

Geometry

Modeling Transformation
Transform into 3D *world* coordinate system

Lighting
Simulate illumination and reflectance

Viewing Transformation
Transform into 3D *camera* coordinate system

Clipping
Clip primitives outside camera’s view

Projection
Transform into 2D camera coordinate system

Scan Conversion
Draw pixels (incl. texturing, hidden surface…)

Image
Computer Graphics Pipeline

Geometry

Modeling Transformation
- Transform into 3D world coordinate system

Lighting
- Simulate illumination and reflectance

Viewing Transformation
- Transform into 3D camera coordinate system

Clipping
- Clip primitives outside camera’s view

Projection
- Transform into 2D camera coordinate system

Scan Conversion
- Draw pixels (incl. texturing, hidden surface…)

Image
Perspective projection

\[ \frac{y}{f} = \frac{Y}{Z} \quad \rightarrow \quad y = f \frac{Y}{Z} \quad \& \quad x = f \frac{X}{Z} \]
Perspective Projection

\[
\begin{pmatrix}
    x \\
    y \\
    fX/Z \\
    fY/Z
\end{pmatrix} = \begin{pmatrix}
    fX/Z \\
    fY/Z
\end{pmatrix} = \begin{pmatrix}
    f & 0 & 0 & 0 \\
    0 & f & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    1 & & & 
\end{pmatrix} \begin{pmatrix}
    X \\
    Y \\
    Z
\end{pmatrix}
\]
Projection Transformations

void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
Projection Transformations

```c
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
```
Projection Transformations

void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
             GLdouble top, GLdouble near, GLdouble far);

void gluOrtho2D(GLdouble left, GLdouble right,
                GLdouble bottom, GLdouble top, GLdouble near, GLdouble far, GLdouble far);
Computer Graphics Pipeline

**Geometry**

- **Modeling Transformation**: Transform into 3D *world* coordinate system
- **Lighting**: Simulate illumination and reflectance
- **Viewing Transformation**: Transform into 3D *camera* coordinate system
- **Clipping**: Clip primitives outside camera’s view
- **Projection**: Transform into 2D camera coordinate system
- **Scan Conversion**: Draw pixels (incl. texturing, hidden surface…)

**Image**
Scan Conversion/Rasterization

- Determine which fragments get generated
- Interpolate parameters (colors, textures, normals, etc.)
Scan Conversion/Rasterization

- Determine which fragments get generated
- Interpolate parameters (colors, textures, normals, etc.)
Scan Conversion/Rasterization

• Determine which fragments get generated
• Interpolate parameters (colors, textures, normals, etc.)

• How?
Scan Conversion/Rasterization

- Determine which fragments get generated
- Interpolate parameters (colors, textures, normals, etc.)

- Barycentric coords amongst many other ways...
Barycentric coordinates

\[ q = \alpha p_1 + \beta p_2 + \gamma p_3 \]

If \([\alpha + \beta + \gamma = 1 \text{ and } \{\alpha, \beta, \gamma\} \geq 0]\), then q inside triangle \((p_1, p_2, p_3)\)

Can also write:

\[ q = \alpha p_1 + \beta p_2 + (1 - \alpha - \beta)p_3 \]
Barycentric coordinates

How to solve for $\alpha$ and $\beta$ in
$q = \alpha p_1 + \beta p_2 + (1 - \alpha - \beta)p_3$?

Two equations, two unknowns:
use 2x2 matrix inversion…
Additional concept: Texture mapping

• Model surface-detail with images
  – wrap object with photograph(s)
  – graphics object itself is a simpler model but “looks” more complex
Texture mapping

• Model surface-detail with images
  – wrap object with photograph(s)
  – graphics object itself is a simpler model but "looks" more complex
Texture coordinates

- Mechanism for attaching the texture map to the surface modeled
  - a pair of floats \((s, t)\) for each triangle vertex
  - corners of the image are \((0, 0), (0, 1), (1, 1), \) and \((1, 0)\)
  - tiling indicated with tex. coords. > 1
  - texels – color samples in texture maps
Texture coordinates
Texture mapping

\[ P_1'(0, 0) \]
\[ P_2'(1, 0) \]
\[ P_3'(1, 1) \]
\[ P_4'(0, 1) \]
Texels: texture elements

\[ P_1'(u_1, v_1, s_1, t_1) \]

\[ P'(u, v, s, t) \]

\[ P_3'(u_3, v_3, s_3, t_3) \]

\[ P_2'(u_2, v_2, s_2, t_2) \]
Problem: how to compute the texture coordinates for an interior pixel?
Texture mapping

Solution: *interpolate vertex texture coordinates*
Parameter Interpolation

• Texture coordinates, colors, normals, etc.

• How?
  – Again, use barycentric coordinates...