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

Fall 2017

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

- **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*
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

Identity

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

Scale

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

Translation

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

Mirror over X axis

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

Rotate around Z axis:
\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
\cos\Theta & -\sin\Theta & 0 & 0 \\
\sin\Theta & \cos\Theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

Rotate around Y axis:
\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
\cos\Theta & 0 & -\sin\Theta & 0 \\
0 & 1 & 0 & 0 \\
\sin\Theta & 0 & \cos\Theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]

And many more…

Rotate around X axis:
\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos\Theta & -\sin\Theta & 0 \\
0 & \sin\Theta & \cos\Theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
w
\end{bmatrix}
\]
Computer Graphics Pipeline

- **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 Scatterring

(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

\[
\tilde{x}_c = R(\tilde{X} - C) \\
\tilde{x}_c = R\tilde{X} - RC \\
\tilde{x}_c = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}
\]

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

\[
t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}^T \\
\text{translation vector}
\]

World-to-camera matrix \( M \)
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

\[ y = \frac{Y}{\frac{f}{Z}} \]

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

\[
\begin{pmatrix}
  x \\
  y \\
  fY/Z
\end{pmatrix} = \begin{pmatrix}
  fX/Z \\
  fY/Z
\end{pmatrix}
\]

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

```c
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
            GLdouble top, GLdouble near, GLdouble far);

void gluOrtho2D(GLdouble left, GLdouble right,
                GLdouble bottom, GLdouble top);
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

If \( \alpha + \beta + \gamma = 1 \) 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 mapping

- Generic image to represent material
  - e.g., tile pattern

bark

veneer

bricks
Tiling

• Repeat pattern
Tiling

• Repeat pattern
Tiling

- Repeat pattern
- How can we improve?
Tiling

• Repeat pattern
  – reduce seems by mirroring
Tiling

• Repeat pattern
  – reduce seems by mirroring
Tiling

- Repeat pattern
  - reduce seems by mirroring
Tiling

- Repeat pattern
  - reduce seems by mirroring
  - How we can further improve?
Tiling

- Repeat pattern
  - reduce seems by mirroring
  - reduce seems by choosing tile that covers one period of repeated texture
Tiling
Texture mapping limitations do exist...
Bricks are similar not identical
Solution?
Solution: Texture synthesis...
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

\[ P_1'(0, 0) \]
\[ P_2'(0, 0) \]
\[ P_3'(0, 1) \]
\[ P_4'(1, 1) \]

\[ P_1(1, 0) \]
\[ P_2(0, 0) \]
\[ P_3(0, 1) \]
\[ P_4(1, 1) \]
Texels: texture elements
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...
Level of detail problem

If curious, you can read more on this subject!