The Road So Far...

Last week

Construct geometry
Define transformations
Assign material properties
...

Vertex Transformations

This week

Culling & Clipping
Determine front-facing triangles
Determine which vertices are visible

Rasterization
Fill the triangle with fragments

Fragment Shading
Calculate correct color values

Visibility Tests
Blending
Is the fragment visible?
Blend together multiple fragments

Vertex Shader
Object's local space → viewport space
Transformations

- Watch the Computerphile video, try to find out:
  1) Why are we using matrices?

The True Power of the Matrix (Transformations in Graphics) – Computerphile
https://www.youtube.com/watch?v=vQ60rFwh2ig
Transformations

• Watch the Computerphile video, try to find out:
  1) Why are we using matrices?
  2) Where do the homogeneous coordinates come in?

The True Power of the Matrix (Transformations in Graphics) – Computerphile
https://www.youtube.com/watch?v=vQ60rFwh2ig
Linear Transformations

- Also called *linear mapping, linear function*
Linear Transformations

- Also called *linear mapping, linear function*

- Transforms a vector space $V$ into a vector space $W$, while preserving addition and scalar multiplication
Linear Transformations

• Also called linear mapping, linear function

• Transforms a vector space $V$ into a vector space $W$, while preserving addition and scalar multiplication

• Satisfies: $f(\alpha \cdot v + \beta \cdot u) = \alpha \cdot f(v) + \beta \cdot f(u)$
Linear Transformations

- Also called *linear mapping*, *linear function*
- Transforms a vector space $V$ into a vector space $W$, while preserving addition and scalar multiplication

- Satisfies: $f(\alpha \cdot v + \beta \cdot u) = \alpha \cdot f(v) + \beta \cdot f(u)$

- In 3D: $\alpha, \beta \in \mathbb{R}$, $u, v \in \mathbb{R}^3$
Linear Transformations

- Take our vector space of points
Linear Transformations

• Take our vector space of points

• Take for example a point \( p = (2, 1) \)
Linear Transformations

- Take our vector space of points
- Take for example a point \( p = (2, 1) \)
- Try mappings:
  1) \( f(p) = (p_x, p_y) \)
  2) \( f(p) = (2 \cdot p_x, p_y) \)
  3) \( f(p) = (p_x, 2 \cdot p_y) \)
  4) \( f(p) = (2 \cdot p_x, 2 \cdot p_y) \)

Test the linearity at home...
Linear Transformations

• From Algebra you know that all linear transformations can be represented as matrices.

Linear transformation $\rightarrow$ Matrix
Linear Transformations

• From Algebra you know that all linear transformations can be represented as matrices.

• Every matrix also gives you a linear transformation.

Linear transformation $\rightarrow$ Matrix

Linear transformation $\leftarrow$ Matrix
Linear Transformations

• What would be the matrices for the linear transformations we just saw?

\[
f(p) = \begin{pmatrix} \_ & \_ \\ \_ & \_ \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix}
\]

\[
f(p) = (p_x, p_y)
\]

\[
f(p) = (2 \cdot p_x, p_y)
\]

\[
f(p) = (p_x, 2 \cdot p_y)
\]

\[
f(p) = (2 \cdot p_x, 2 \cdot p_y)
\]
Scale

- Stretches or shrinks the space

\[
\begin{align*}
\text{2D} & : \begin{pmatrix} a_x & 0 \\ 0 & a_y \end{pmatrix} & a_x \text{ – x-axis scale factor} \\
\text{3D} & : \begin{pmatrix} a_x & 0 & 0 \\ 0 & a_y & 0 \\ 0 & 0 & a_z \end{pmatrix} & a_x \text{ – x-axis scale factor} \\
& & a_y \text{ – y-axis scale factor} \\
& & a_z \text{ – z-axis scale factor}
\end{align*}
\]
Scale

- Transformations can be easily understood, if we see what they do with the standard basis.
Scale

- Transformations can be easily understood, if we see what they do with the standard basis.
Scale

- Transformations can be easily understood, if we see what they do with the standard basis.

- Furthermore, one can read the transformed standard basis from the columns of the transformation matrix!
Shear

- Shear-x, shear-y
- Tilts one of the axes parallel to other(s)
Shear

- **Shear-y**, we tilt the $x$ basis vector parallel to $y$ by angle $\phi$ counterclockwise
  \[
  \begin{pmatrix}
  1 & 0 \\
  \tan(\varphi) & 1
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  =
  \begin{pmatrix}
  x \\
  y + \tan(\varphi) \cdot x
  \end{pmatrix}
  \]

- **Shear-x**, we tilt the $y$ basis vector parallel to $x$ by angle $\phi$ clockwise
  \[
  \begin{pmatrix}
  1 & \tan(\varphi) \\
  0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  =
  \begin{pmatrix}
  x + \tan(\varphi) \cdot y \\
  y
  \end{pmatrix}
  \]

What about in 3D?
Rotation

- We want to keep the basis vectors on the unit-circle.

Can you derive the matrix?
Rotation

\[ e'_0 = (|a|, |b|) = (\cos(\alpha), \sin(\alpha)) \]
\[ e'_1 = (|a'|, |b'|) = (-\sin(\alpha), \cos(\alpha)) \]

\[ \cos(\alpha) = \frac{|a|}{|e'_0|} = \frac{|a|}{1} = |a| \]
Rotation

- Rotates around the $z$ axis by the angle $\alpha$

$$2D \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

$\alpha$ – Positive angle to rotate by
Rotation

- Rotates around the $z$ axis by the angle $\alpha$

\[
\begin{pmatrix}
\cos(\alpha) & -\sin(\alpha) \\
\sin(\alpha) & \cos(\alpha)
\end{pmatrix}
\]

- Similar matrices that rotate around each main axis.

$\alpha$ – Positive angle to rotate by
Rotation

- Rotates around the $z$ axis by the angle $\alpha$

\[
\begin{pmatrix}
\cos(\alpha) & -\sin(\alpha) \\
\sin(\alpha) & \cos(\alpha)
\end{pmatrix}
\]

$\alpha$ – Positive angle to rotate by

- Similar matrices that rotate around each main axis.

- What about rotation around an arbitrary axis?
Linear Transformations

Defined geometry
Linear Transformations
Linear Transformations

Scale
Linear Transformations
Linear Transformations

Rotation
Linear Transformations
Linear Transformations

Shear
Linear Transformations

Will these be enough?
Translation

- Imagine a 1D world located at $y=1$ line in 2D.
Translation

- Imagine a 1D world located at $y=1$ line in 2D.
Translation

- Imagine a 1D world located at $y=1$ line in 2D.

- Notice that all the points are in the form: $(x, 1)$
Translation

- How to transform the 2D space so that stuff in the 1D hyperplane $y=1$ moves an equal amount?
Translation

• Shear-x by $\tan(45^\circ) = 1$

• Shear-x with $\tan(63.4^\circ) = 2$
Translation

- Affine transformation in the current space, linear shear transformation in 1 dimension higher space.

\[
\begin{align*}
\text{2D} & \quad \begin{pmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_t \\ y + y_t \\ 1 \end{pmatrix} \\
\text{Shear-xy} & \\
\text{3D} & \quad \begin{pmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_t \\ y + y_t \\ z + z_t \\ 1 \end{pmatrix} \\
\text{Shear-xyz} & \\
\text{1D} & \quad \begin{pmatrix} 1 & x_t \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_t \\ 1 \end{pmatrix} \\
\text{Shear-x} &
\end{align*}
\]
Transformations

- This together gives us a very good toolset to transform our geometry as we wish.

\[
\begin{bmatrix}
a & b & c & x_t \\
d & e & f & y_t \\
g & h & i & z_t \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix} =
\begin{pmatrix}
ax + by + cz + x_t \\
dx + ey + fz + y_t \\
gx + hy + iz + z_t \\
1
\end{pmatrix}
\]
Transformations

- This together gives us a very good toolset to transform our geometry as we wish.

**Linear transformations**

\[
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i \\
  0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
  x_t \\
  y_t \\
  z_t \\
  1
\end{bmatrix}
= 
\begin{bmatrix}
  ax + by + cz + x_t \\
  dx + ey + fz + y_t \\
  gx + hy + iz + z_t \\
  1
\end{bmatrix}
\]
Transformations

This together gives us a **very good toolset** to transform our geometry as we wish.

\[
\begin{pmatrix}
    a & b & c & x_t \\
    d & e & f & y_t \\
    g & h & i & z_t \\
    0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    x \\
    y \\
    z \\
    1
\end{pmatrix} =
\begin{pmatrix}
    ax + by + cz + x_t \\
    dx + ey + fz + y_t \\
    gx + hy + iz + z_t \\
    1
\end{pmatrix}
\]
Transformations

- This together gives us a very good toolset to transform our geometry as we wish.

\[
\begin{bmatrix}
 a & b & c \\
 d & e & f \\
 g & h & i \\
 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
 x_t \\
 y_t \\
 z_t \\
 1 \\
\end{bmatrix}
= 
\begin{bmatrix}
 ax + by + cz + x_t \\
 dx + ey + fz + y_t \\
 gx + hy + iz + z_t \\
 1 \\
\end{bmatrix}
\]
Multiple Transformations

• How can we apply multiple transformations?

\[ A \cdot (B \cdot (C \cdot v)) \]

• Is it the same as?

\[ B \cdot (A \cdot (C \cdot v)) \]
Transformations

- In some graphics libraries you assign the position / translation, rotation and scale individually.
Transformations

- In some graphics libraries you assign the \texttt{position / translation, rotation} and \texttt{scale} individually.

```
object.position.set(2.7, 1.2, 0);
object.scale.set(2.4, 0.1, 0.4);
object.rotation.set(0, toRad(180), 0);
```
Transformations

• In some graphics libraries you assign the position / translation, rotation and scale individually.

• To the GPU the object transformations are sent as a matrix (model matrix).
Transformations

- In some graphics libraries you assign the position / translation, rotation and scale individually.
- To the GPU the object transformations are sent as a matrix (model matrix).

\[
\text{projectionMatrix} \cdot \text{viewMatrix} \cdot \text{modelMatrix} \cdot \mathbf{v}
\]

\[
P \cdot V \cdot M \cdot \mathbf{v}
\]
Transformations

- In some graphics libraries you assign the position / translation, rotation and scale individually.
- To the GPU the object transformations are sent as a matrix \( \text{(model matrix)} \).
- Questions about transformations?
Scene Graph

- Dependency between (parts of) objects.
Scene Graph

Head $S\cdot H \cdot v$

Body $S\cdot B \cdot v$

Left hand $S\cdot B\cdot L \cdot v$

Right hand $S\cdot B\cdot R \cdot v$
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

1) $M = \text{Identity}$, push($M$)
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

1) \( M = \text{Identity}, \) push(\( M \))

2) \( M *= S, \) push(\( M \))  Move to snowman's space
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)

- *Current state* is in the **top of the stack**

1) $M = \text{Identity, push}(M)$
2) $M *= S, \text{push}(M)$
3) $M *= H, \text{push}(M)$  Move to head's space
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

1) $M = \text{Identity}, \ push(M)$
2) $M *= S, \ push(M)$
3) $M *= H, \ push(M)$
4) *Draw head vertices*
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)

- *Current state* is in the **top of the stack**

1. $M = \text{Identity, push}(M)$
2. $M *= S, \text{push}(M)$
3. $M *= H, \text{push}(M)$
4. *Draw head vertices*

We now want to get back to the snowman's space
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

1) $M = \text{Identity}$, push($M$)
2) $M *= S$, push($M$)
3) $M *= H$, push($M$)
4) *Draw head vertices*
5) pop(), $M = \text{top()}$
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

2) ...
3) $M \ast= H$, push(M)
4) *Draw head vertices*
5) pop(), $M = \text{top}()$
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

2) ...
3) $M *= H$, push($M$)
4) *Draw head vertices*
5) pop(), $M = \text{top}()$
6) $M *= B$, push($M$)  Move to body's space
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

2) ...
3) $M \ast= H$, push($M$)
4) *Draw head vertices*
5) pop(), $M = \text{top}()$
6) $M \ast= B$, push($M$)
7) *Draw body vertices*
Matrix Stack

- Stack can be used to save and load matrices (intermediary states)
- *Current state* is in the **top of the stack**

5) …
6) $M *= B$, push($M$)
7) *Draw body vertices*
8) … ?
Matrix Stack

- Each (part of an) **object** can be modelled in its own **local space**.

![Diagram of a cube with labeled vertices showing local space coordinates.](image)
Matrix Stack

- Each (part of an) **object** can be modelled in its own **local space**.

- When we traverse the scene graph, important intermediary states are saved / loaded.
Matrix Stack

- Each (part of an) **object** can be modelled in its own **local space**.
- When we traverse the scene graph, important intermediary states can saved / loaded.
- No need to recalculate same matrix multiplications many times or find inverse transformations.

\[
M = A \cdot B \cdot D \cdot D^{-1} = A \cdot B
\]

vs

```
stack.pop(), M = stack.top()
```
Matrix Stack

- Each (part of an) **object** can be modelled in its own **local space**.
- When we traverse the scene graph, important intermediary states can saved / loaded.
- No need to recalculate same matrix multiplications many times or find inverse transformations.
- Questions about the matrix stack?
What new did you find out today?

What more would you like to know?

Next time

Frames of reference, projections