



GPU PROGRAMMING FOR VIDEO GAMES

3D Coordinates & Transformations

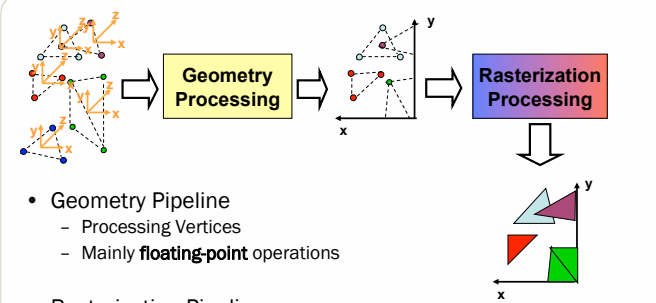


Prof. Aaron Lanterman
(Based on slides by Prof. Hsien-Hsin Sean Lee)
School of Electrical and Computer Engineering
Georgia Institute of Technology



Georgia Institute of Technology

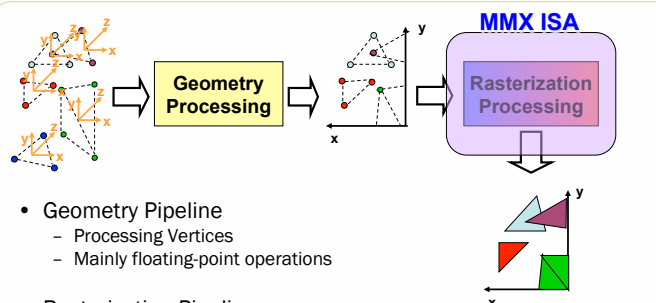
3D graphics rendering pipeline (1)



- Geometry Pipeline
 - Processing Vertices
 - Mainly **floating-point** operations
- Rasterization Pipeline
 - Processing Pixels
 - Mainly dealing with **Integer** operations

Georgia Institute of Technology

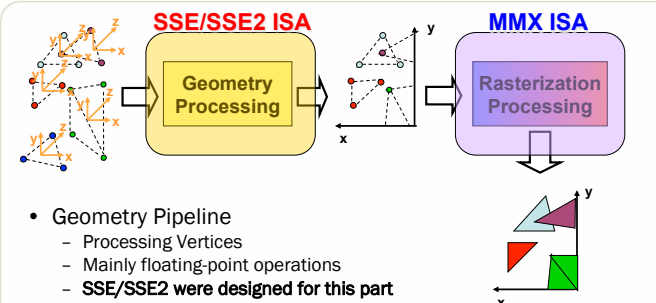
3D graphics rendering pipeline (2)



- Geometry Pipeline
 - Processing Vertices
 - Mainly floating-point operations
- Rasterization Pipeline
 - Processing Pixels
 - Mainly dealing with Integer operations
 - **MMX was originally designed to accelerate this type of functionality**

Georgia Institute of Technology

3D graphics rendering pipeline (3)



- Geometry Pipeline
 - Processing Vertices
 - Mainly floating-point operations
 - **SSE/SSE2 were designed for this part**
- Rasterization Pipeline
 - Processing Pixels
 - Mainly dealing with Integer operations
 - **MMX was originally designed to accelerate this type of functionality**

Georgia Institute of Technology

Fixed-function 3D graphics pipeline

Performed by GPU

- Geometry Pipeline
 - Processing Vertices
 - Mainly floating-point operations
 - ~~SSE/SSE2 were designed for this part~~
- Rasterization Pipeline
 - Processing Pixels
 - Mainly dealing with Integer operations
 - ~~MMX was originally designed to accelerate this type of functionality~~

Georgia Institute of Tech

3D Coord: Math textbooks use z-up

Z-up, Right-Handed System

Georgia Institute of Tech

3D Coord: Real games tend to use y-up

Left-Handed System

- Direct3D
- Unity3D

Right-Handed System

- OpenGL
- XNA

Georgia Institute of Tech

X-Y natural for screen coordinates

Left-Handed System

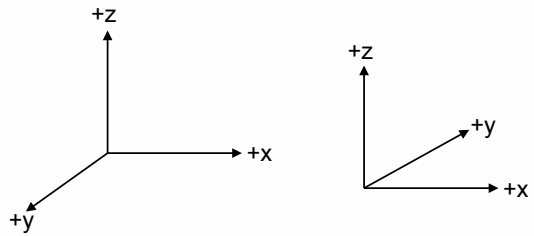
- Direct3D
- Unity3D

Right-Handed System

- OpenGL
- XNA

Georgia Institute of Tech

Some use Z-up for world coordinates

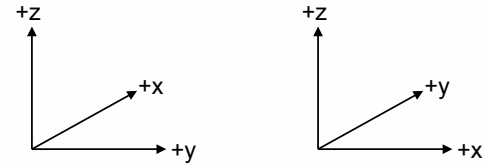


Left-Handed System

Right-Handed System

- Z-up, LHS: Unreal
- Z-up, RHS: Quake/Radiant, Source/Hammer, C4 Engine
- Nearly everything still uses Y-up for screen coordinates!

Another view

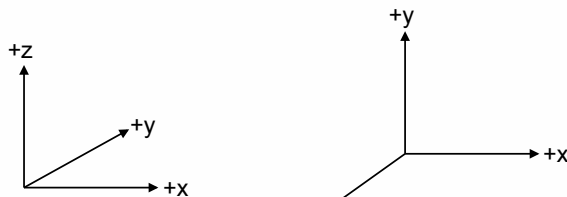


Left-Handed System

Right-Handed System

- Unreal
- Quake/Radiant
- Source/Hammer
- C4 Engine

3D “object” modeling software



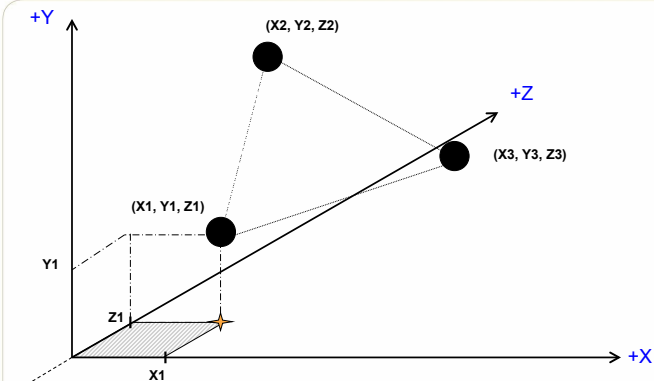
Right-Handed System

3D Studio Max, Blender

Right-Handed System

Maya, Milkshape

Geometry format – vertex coordinates



Geometry format – vertex normals

$(NX1, NY1, NZ1)$
 $(NX2, NY2, NZ2)$
 $(NX3, NY3, NZ3)$

Georgia Institute of Technology

Geometry format – vertex colors

$(R1, G1, B1, A1)$
 $(R2, G2, B2, A2)$
 $(R3, B3, B3, G3)$

Georgia Institute of Technology

Triangle-based geometry representation

Triangle List
(note the vertex order)

Triangle Strip

Triangle Fan

Careful!

Georgia Institute of Technology

Specifying a 3D object (1)

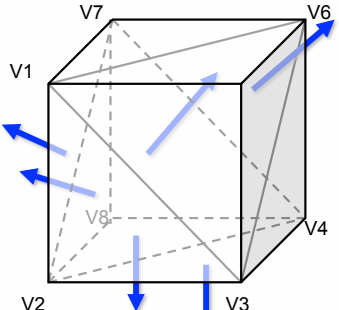
Triangle list
 $\{v1, v3, v2\}$,
 $\{v1, v5, v3\}$,
 $\{v5, v6, v3\}$,
 $\{v4, v3, v6\}$,
 $\{v1, v7, v6\}$,
 $\{v1, v6, v5\}$

Triangle strip
 $\{v5, v3, v1, v2\}$,
 $\{v5, v6, v3, v4\}$,
 $\{v7, v6, v1, v5\}$

- Vertex ordering is critical when culling mode enabled
- We will discuss normal computation later

Georgia Institute of Technology

Specifying a 3D object (2)



Triangle list

- {v1, v2, v7},
- {v2, v8, v7},
- {v2, v3, v4},
- {v2, v4, v8},
- {v4, v7, v8},
- {v4, v6, v7}

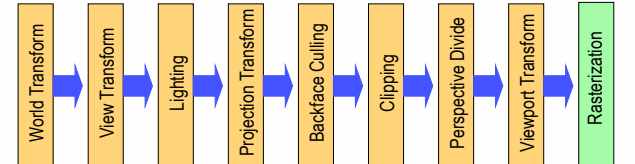
Triangle strip

- {v1, v2, v7, v8},
- {v3, v4, v2, v8},
- {v6, v7, v4, v8}

- Vertex ordering is critical when culling mode enabled
- We will discuss normal computation later

Georgia Institute of Technology

3D rendering pipeline



Georgia Institute of Technology

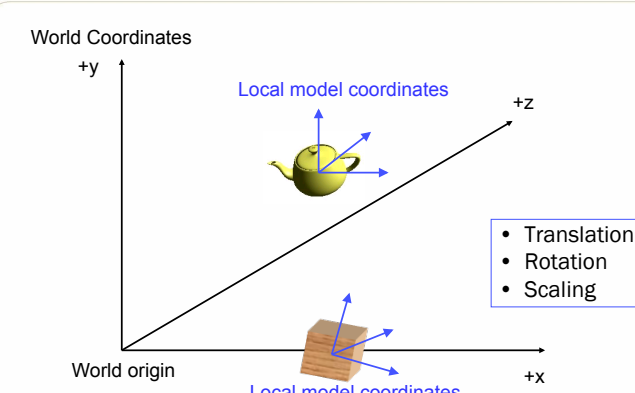
Transformation pipeline

- World Transformation
 - Model coordinates → World coordinates
- View Transformation
 - World coordinates → Camera space
- Projection Transformation
 - Camera space → View plane

• These are a series of matrix multiplications

Georgia Institute of Technology

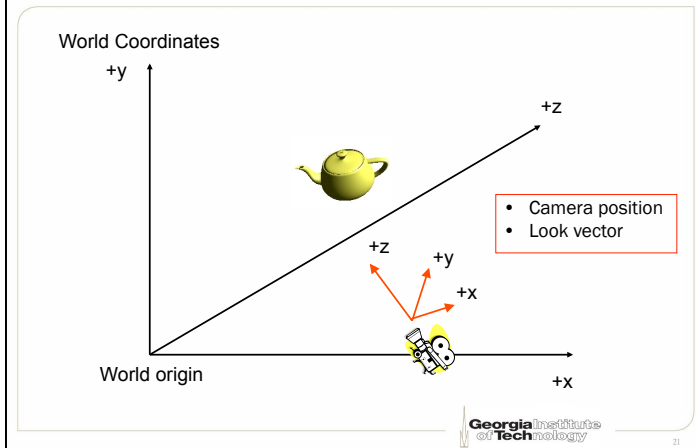
World transformation



- Translation
- Rotation
- Scaling

Georgia Institute of Technology

View transformation



Projection transformation

- Set up camera internals
 - Field of View (FOV)
 - View frustum
 - View planes
- Will discuss in the next lecture

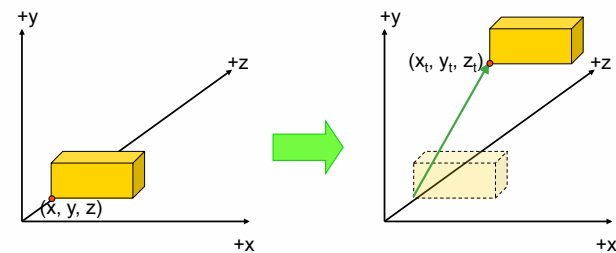
Georgia Institute of Technology

Homogeneous coordinates

- Enable all transformations to be done by “multiplication”
 - Primarily for translation (see next few slides)
- Add one coordinate (w) to a 3D vector
- Each vertex has $[x, y, z, w]$
 - w will be useful for perspective projection
 - w should be 1 in a Cartesian coordinate system

Georgia Institute of Technology

Transformation 1: translation (Offset)

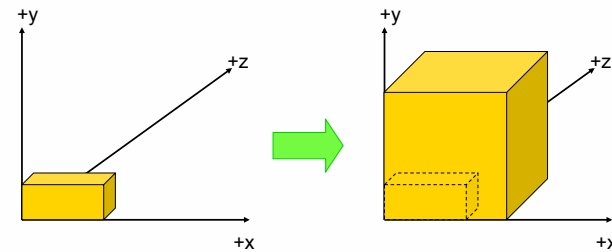


Translation matrix

$$[x_t, y_t, z_t, 1] = [x, y, z, 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

- Example of a row-coordinate convention
- Direct3D, XNA, HLSL/Cg use row coordinates
- OpenGL & non-graphics world uses column coordinates

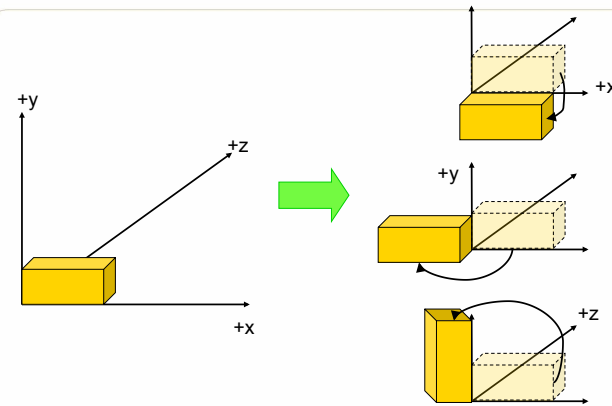
Transformation 2: scaling



Scaling matrix

$$[x_s, y_s, z_s, 1] = [x, y, z, 1] \cdot \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation 3: rotation



2D rotation

$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Rotate along which axis?

Georgia Institute of Technology

3D rotation matrix (LHS)

Rotation along **Z** axis $[x', y', z', 1] = [x, y, z, 1] \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotation along **Y** axis $[x', y', z', 1] = [x, y, z, 1] \cdot \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotation along **X** axis $[x', y', z', 1] = [x, y, z, 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Georgia Institute of Technology

Non-commutative property (1)

1. Counter-clockwise 90° along y
2. Clockwise 90° along x

1. Clockwise 90° along x
2. Counter-clockwise 90° along y

Georgia Institute of Technology

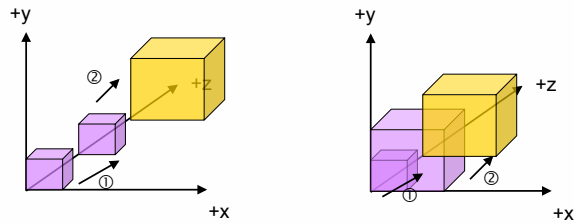
Non-commutative property (2)

- ① $\begin{bmatrix} \cos(-\frac{\pi}{2}) & 0 & -\sin(-\frac{\pi}{2}) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(-\frac{\pi}{2}) & 0 & \cos(-\frac{\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2}) & \sin(\frac{\pi}{2}) & 0 \\ 0 & -\sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- ② $(x'', y'', z'') = (-z, -x, y)$

- ① $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\frac{\pi}{2}) & \sin(\frac{\pi}{2}) & 0 \\ 0 & -\sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(-\frac{\pi}{2}) & 0 & -\sin(-\frac{\pi}{2}) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(-\frac{\pi}{2}) & 0 & \cos(-\frac{\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- ② $(x'', y'', z'') = (-y, -z, x)$

Georgia Institute of Technology

Non-commutative property (3)



1. Translation by (x, y, z)
2. Scale by 2 times

1. Scale by 2 times
2. Translation by (x, y, z)

Non-commutative property (4)

$$\begin{matrix} \textcircled{1} & \textcircled{2} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} & \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & = & \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ S_x T_x & S_y T_y & S_z T_z & 1 \end{bmatrix} \end{matrix}$$

$(x'', y'', z'') = (x * S_x + S_x * T_x, y * S_y + S_y * T_y, z * S_z + S_z * T_z)$

Offsets were scaled as well

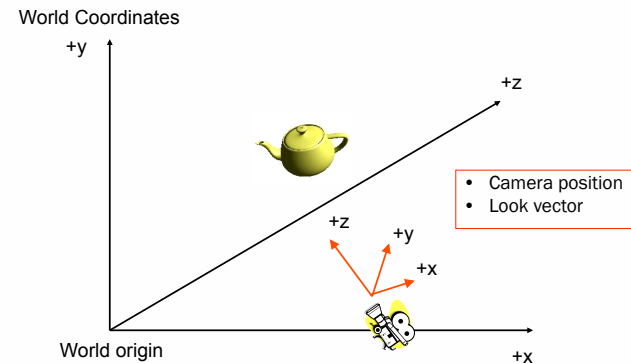
$$\begin{matrix} \textcircled{1} & \textcircled{2} \\ \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} & = & \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} \end{matrix}$$

$(x'', y'', z'') = (x * S_x + T_x, y * S_y + T_y, z * S_z + T_z)$

Non-commutative property (5)

- Ordering matters !
- Be careful when performing matrix multiplication

View transformation revisited



Specifying the view transformation

- Most commonly parameterized by:
 - Position of camera
 - Position of point to look at
 - Vector indicating “up” direction of camera
- In Direct3D: `D3DXMatrixLookAtLH`
 - D3D uses a LHS, but also have `D3DXMatrixLookAtRH`
- In XNA: `Matrix.CreateLookAt` (RHS)
- In OpenGL: `gluLookAt` (RHS)
- Can also build a rotation+translation matrix as if the camera was an object in scene, then take the inverse of that matrix

[msdn.microsoft.com/en-us/library/bb205342\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb205342(VS.85).aspx)
[msdn.microsoft.com/en-us/library/bb205343\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb205343(VS.85).aspx)

Georgia Institute
of Technology