Review: Computer Graphics Basics

1

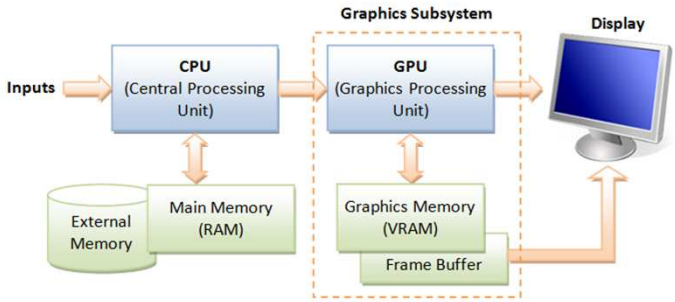
Computer Graphics Definition

- Remember that Computer Graphics is
  - Producing pictures or images using a computer
  - Generating 2D images of a 3D world represented in a computer

2

Computer Graphics Hardware

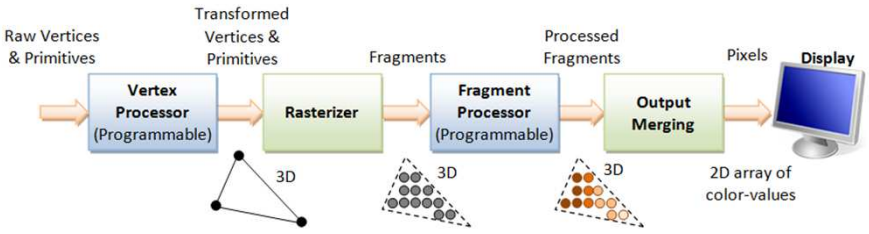
- Computers have *Graphics Processing Unit (GPU)* to produce images for the display, with its own graphics memory (or Video RAM or VRAM)
- The color values of the pixels are stored in a special part of graphics memory called *frame buffer*



3

3D Graphics Rendering Pipeline

- *Rendering* is the process of producing image on the display from model description



3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

4

3D Graphics Rendering Pipeline

- The 3D graphics rendering pipeline consists of the following main stages:
  1. Vertex Processing: Process and transform individual vertices
  2. Rasterization: Convert each primitive into a set of fragments
  3. Fragment Processing: Process individual fragments
  4. Output Merging: Combine the fragments of all primitives (in 3D space) into 2D color-pixel for the display

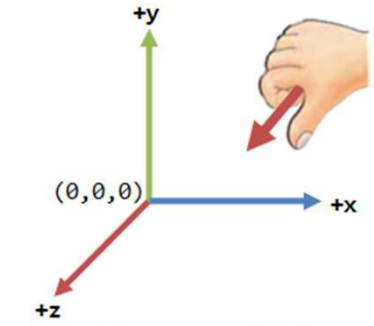
5

3D Coordinates System

- Coordinate system: has an origin and some mutually perpendicular axes emanating from the origin
- In Right-Hand System (RHS), the x-axis is pointing right, y-axis is pointing up, and z-axis is pointing out of the screen
- With your right-hand fingers curving from the x-axis towards the y-axis, the thumb is pointing at the z-axis. RHS is *counter-clockwise* (CCW)
- OpenGL uses Right-Hand System
- Microsoft Direct3D uses Left-Hand System

6

3D Coordinates System

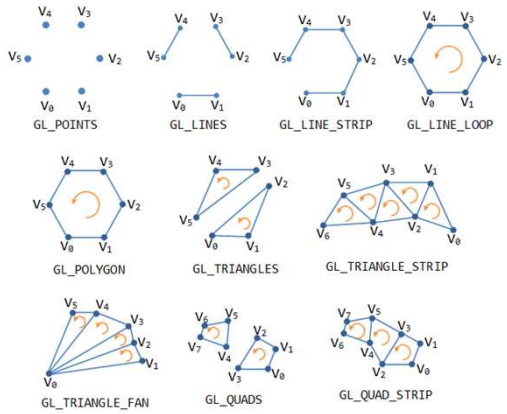


Right-Hand System (RHS) or Counter-Clockwise (CCW) System

7

Primitives

- The inputs to the Graphics Rendering Pipeline are geometric *primitives* (such as triangle, point, line or quad), which is formed by *one or more vertices*



OpenGL Primitives

8

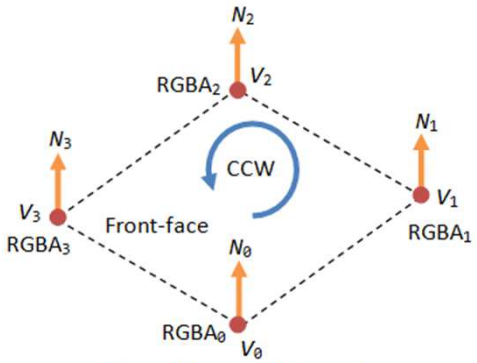
Vertices

- A *vertex*, in computer graphics, has these attributes:

1. Position in 3D space $V=(x, y, z)$
2. Color: expressed in RGB (Red-Green-Blue) or RGBA (Red-Green-Blue-Alpha)
3. Vertex-Normal $N=(n_x, n_y, n_z)$: the normal vector is perpendicular to the surface.
4. Texture $T=(s, t)$: In computer graphics, we often wrap a 2D image to an object to make it seen realistic.
5. Others.

9

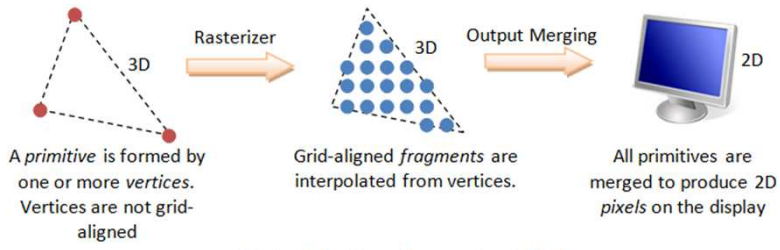
Vertices



A quad formed by 4 vertices (with color and normal)

10

Pixel vs. Fragment



Vertex, Primitives, Fragment and Pixel

11

Pixel vs. Fragment

- Pixels refer to the dots on the display, which are aligned in a 2-dimensional grid of a certain rows and columns
  - A pixel is 2-dimensional, with a (*x*, *y*) position and a RGB color value (there is no alpha value for pixels)
- The *rasterizer* takes each input primitive and perform *raster-scan* to produce a set of grid-aligned fragments enclosed within the primitive.
  - A fragment is 3-dimensional, with a (*x*, *y*, *z*) position. The (*x*, *y*) are aligned with the 2D pixel-grid. The *z*-value (not grid-aligned) denotes its depth.
  - The z-values are needed to capture the relative depth of various primitives, so that the occluded objects can be discarded

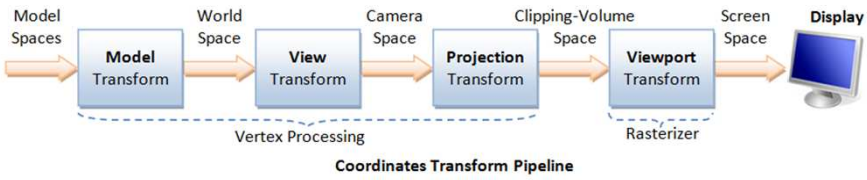
12

3D Graphics Rendering Pipeline

- The 3D graphics rendering pipeline consists of the following main stages:
  1. **Vertex Processing: Process and transform individual vertices**
  2. Rasterization: Convert each primitive into a set of fragments
  3. Fragment Processing: Process individual fragments
  4. Output Merging: Combine the fragments of all primitives (in 3D space) into 2D color-pixel for the display

13

Coordinates Transformation



Coordinates Transform Pipeline

14

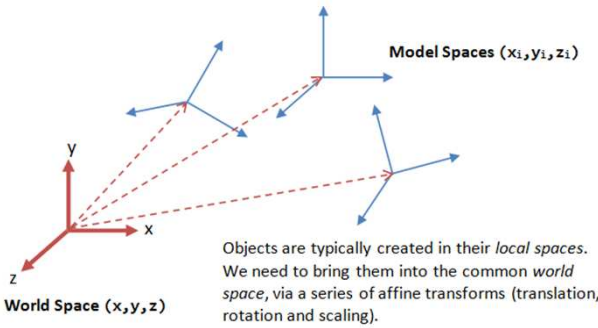
Coordinates Transformation

- The process involves four transformations:

1. Arrange the objects in the world (*Model Transformation* or *World transformation*).
2. Position and orientation the camera (*View transformation*).
3. Select a camera lens (wide angle, normal or telescopic), adjust the focus length and zoom factor to set the camera's field of view (*Projection transformation*).
4. Print the photo on a selected area of the paper (*Viewport transformation*)

15

Model Transform



Objects are typically created in their *local spaces*. We need to bring them into the common *world space*, via a series of affine transforms (translation, rotation and scaling).

16

Model Transform

- Each object in a 3D scene is typically drawn in its own coordinate system, known as its *model space*
- We need to transform the vertices from their local spaces to the *world space*
- The transform consists of a series of
  - Scaling, rotation, and translation
- A transform converts a vertex *V* from one space (or coordinate system) to another space *V'* is carried by multiplying the vector with a *transformation matrix*, i.e., ***V' = MV***

17

Model Transform

- In homogenous coordinates, a vertex ***V*** at (*x*, *y*, *z*) is represented as a 4x1 column vector

$$V = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Scaling can be represented in a 4x4 matrix

$$S(\alpha_x, \alpha_y, \alpha_z) = \begin{bmatrix} \alpha_x & 0 & 0 & 0 \\ 0 & \alpha_y & 0 & 0 \\ 0 & 0 & \alpha_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- where α_x, α_y and α_z represent the scaling factors in *x*, *y* and *z* direction

18

Model Transform

$$V' = SV = \begin{bmatrix} \alpha_x & 0 & 0 & 0 \\ 0 & \alpha_y & 0 & 0 \\ 0 & 0 & \alpha_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 3D Rotations about the *x*, *y* and *z* axes for an angle θ (measured in counter-clockwise manner) can be represented:

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

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

19

Model Transform

- Translation can be represented in a 4x4 matrix, as follows

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

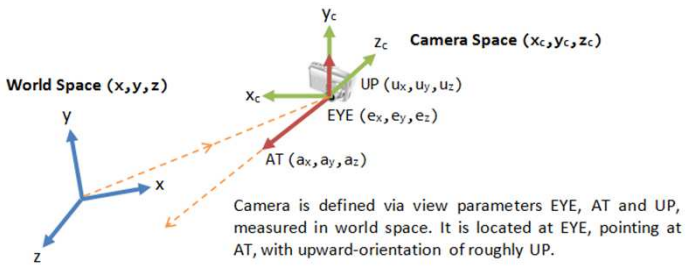
- A series of successive affine transforms ($\boldsymbol{T_1}$, $\boldsymbol{T_2}$, $\boldsymbol{T_3}$, ...) can be computed via concatenated matrix multiplications $\boldsymbol{V' = ...T_3T_2T_1V}$

- The matrices can be combined before applying to the vertex because matrix multiplication is associative

$$\boldsymbol{T_3\,(T_2\,(T_1\,V)\,)} = \boldsymbol{(\,T_3T_2T_1\,)\,V}$$

20

View Transform



Camera is defined via view parameters EYE, AT and UP, measured in world space. It is located at EYE, pointing at AT, with upward-orientation of roughly UP.

In the Camera space, camera is located at origin, pointing at $-z_c$, with upward-orientation of y_c.
z_c is opposite of AT, y_c is roughly UP.

21

View Transform

- In 3D graphics, we position the camera onto the world space by specifying three *view parameters*: EYE, AT and UP, in world space
  1. The point EYE (e_x, e_y, e_z) defines the location of the camera
  2. The vector AT (a_x, a_y, a_z) denotes the direction where the camera is aiming at, usually at the center of the world or an object
  3. The vector UP (u_x, u_y, u_z) denotes the upward orientation of the camera roughly
- Camera coordinates computed as:

$$z_c = \frac{EYE - AT}{\|EYE - AT\|}$$
$$x_c = \frac{UP \times z_c}{\|UP \times z_c\|}$$
$$y_c = z_c \times x_c$$

22

From World Space to Camera Space

- World space is represented by orthonormal bases (e_1, e_2, e_3), where e_1=(1, 0, 0), e_2=(0, 1, 0) and e_3=(0, 0, 1), with origin at O=(0, 0, 0)
- The camera space has orthonormal bases (x_c, y_c, z_c) with origin at EYE=(e_x, e_y, e_z)

$$T(-EYE) = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} {x^c}_x & {x^c}_y & {x^c}_z & 0 \\ {y^c}_x & {y^c}_y & {y^c}_z & 0 \\ {z^c}_x & {z^c}_y & {z^c}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ where } x_c = \begin{bmatrix} {x^c}_x \\ {x^c}_y \\ {x^c}_z \end{bmatrix}, y_c = \begin{bmatrix} {y^c}_x \\ {y^c}_y \\ {y^c}_z \end{bmatrix}, \text{ and } z_c = \begin{bmatrix} {z^c}_x \\ {z^c}_y \\ {z^c}_z \end{bmatrix}$$

23

From World Space to Camera Space

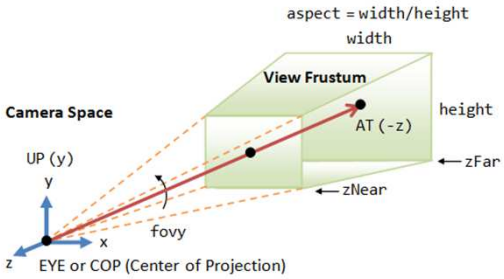
- We can combine the two operations into one single *View Matrix*:

$$M_{view} = T.R = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} {x^c}_x & {x^c}_y & {x^c}_z & 0 \\ {y^c}_x & {y^c}_y & {y^c}_z & 0 \\ {z^c}_x & {z^c}_y & {z^c}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

24

Projection Transform - Perspective

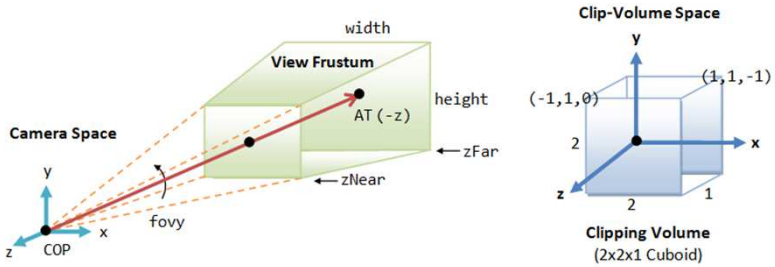
- The projection with view frustum is known as *perspective projection*
  - objects nearer to the COP (Center of Projection) appear larger than objects further to the COP of the same size.
- An object outside the view frustum is not visible to the camera and shall be discarded. This is known as *view-frustum culling*



Perspective Projection: The camera's view frustum is specified via 4 view parameters: fovy, aspect, zNear and zFar.

Clipping-Volume Cuboid

- Projection matrix transforms the view-frustum into a axis-aligned cuboid clipping-volume of 2x2x1 centered on the near plane
- The near plane has z=0, whereas the far plane has z=-1. The planes have dimension of 2x2, with range from -1 to +1



Perspective Projection Matrix

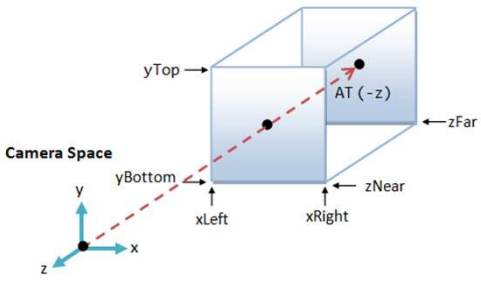
- $M_{proj} = \begin{bmatrix} \frac{\cot(\frac{fovy}{2})}{aspect} & 0 & 0 & 0 \\ 0 & \cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & \frac{zFar}{zFar - zNear} & \frac{zNear * zFar}{zFar - zNear} \\ 0 & 0 & -1 & 0 \end{bmatrix}$

- With input vertex of (x, y, z, 1), the resultant *w*-component would not be 1. We need to normalize the resultant homogeneous coordinates (x, y, z, w) to (x/w, y/w, z/w, 1) to obtain position in 3D space.

27

Projection Transform - Orthographic

- A special case where the camera is placed very far away from the world (analogous to using telescopic lens).
- The view volume for orthographic projection is a *parallelepiped* (instead of a frustum in perspective projection).



Orthographic Projection: Camera positioned infinitely far away at $z = \infty$

28

Output of Vertex Processing

- Each vertex is transformed and positioned in the clipping-volume cuboid space, together with their vertex-normal
- The x and y coordinates (in the range of -1 to +1) represent its position on the screen, and the z value (in the range of 0 to 1) represents its depth, i.e., how far away from the near plane

29

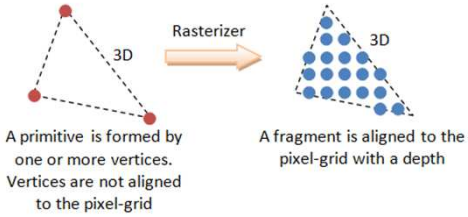
3D Graphics Rendering Pipeline

- The 3D graphics rendering pipeline consists of the following main stages:
  1. Vertex Processing: Process and transform individual vertices
  2. **Rasterization: Convert each primitive into a set of fragments**
  3. Fragment Processing: Process individual fragments
  4. Output Merging: Combine the fragments of all primitives (in 3D space) into 2D color-pixel for the display

30

Rastarization

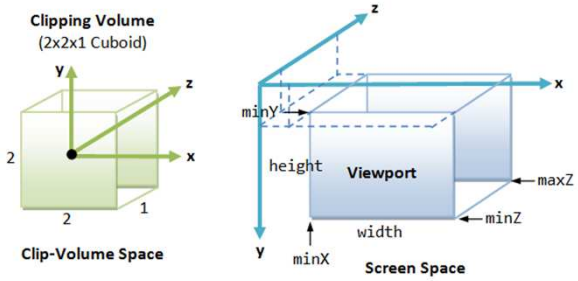
- Primitive (such as triangle, quad, point and line), which is defined by one or more vertices, are *raster-scan* to obtain a set of fragments enclosed within the primitive
- The 3D fragments, which are *interpolated* from the vertices, have the same set of attributes as the vertices, such as position, color, normal, texture



31

Viewport Transform

- Viewport is a rectangular display *area* on the application window, which is measured in screen's coordinates (in pixels, with origin at the top-left corner)
- In 3D graphics, a viewport is 3-dimensional to support z-ordering, which is needed for situations such as ordering of overlapping windows.



32

Viewport Transform

- Reflection of y-axis $M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- Scaling of x, y, and z axes $M_2 = \begin{bmatrix} w/2 & 0 & 0 & 0 \\ 0 & h/2 & 0 & 0 \\ 0 & 0 & maxZ - minZ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- Translation of $origin$ $M_3 = \begin{bmatrix} 1 & 0 & 0 & minX + w/2 \\ 0 & 1 & 0 & minY + h/2 \\ 0 & 0 & 1 & minZ \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- $M_{viewport} = M_3 M_2 M_1 = \begin{bmatrix} w/2 & 0 & 0 & minX + w/2 \\ 0 & -h/2 & 0 & minY + h/2 \\ 0 & 0 & maxZ - minZ & minZ \\ 0 & 0 & 0 & 1 \end{bmatrix}$

33

Back Face Culling

- While view frustum culling discard objects outside the view frustum, back-face culling discard primitives which is not facing the camera
- Back face can be declared based on the normal vector and the vector connecting the surface and the camera
- Back-face culling shall not be enabled if the object is transparent and alpha blending is enabled

34

3D Graphics Rendering Pipeline

- The 3D graphics rendering pipeline consists of the following main stages:
  1. Vertex Processing: Process and transform individual vertices
  2. Rasterization: Convert each primitive into a set of fragments
  3. **Fragment Processing: Process individual fragments**
  4. Output Merging: Combine the fragments of all primitives (in 3D space) into 2D color-pixel for the display

35

Fragment Processing

- The fragment processing focuses on the *texture* and *lighting*, which has the greatest impact on the quality of the final image

- The operations involved in the fragment processor are:
  1. The first operation in fragment processing is texturing
  2. Next, primary and secondary colors are combined, and fog calculation may be applied
  3. The optional scissor test, alpha test, stencil test, and depth-buffer test are carried out, if enabled
  4. Then, the optional blending, dithering, logical operation, and bitmasking may be performed

36

3D Graphics Rendering Pipeline

- The 3D graphics rendering pipeline consists of the following main stages:
  1. Vertex Processing: Process and transform individual vertices
  2. Rasterization: Convert each primitive into a set of fragments
  3. Fragment Processing: Process individual fragments
  4. **Output Merging: Combine the fragments of all primitives (in 3D space) into 2D color-pixel for the display**

37

Z-Buffer and Hidden-Surface Removal

- *z-buffer* (or *depth-buffer*) can be used to remove hidden surfaces (surfaces blocked by other surfaces and cannot be seen from the camera)
- The z-buffer of the screen is initialized to 1 (farthest) and color-buffer initialized to the background color
- For each fragment, its z-value is checked against the buffer value:
  - If its z-value is smaller than the z-buffer, its color and z-value are copied into the buffer
  - Otherwise, this fragment is occluded by another object and discarded

38

Alpha Blending

- A fragment is not necessarily opaque, and could contain an alpha value specifying its degree of transparency
- The alpha is typically normalized to the range of [0, 1], with 0 denotes totally transparent and 1 denotes totally opaque
- If the fragment is not totally opaque, then part of its background object could show through, which is known as *alpha blending*

- The simplest blending equation is as follows:

$$c = \alpha_s c_s + (1 - \alpha_s) c_d$$

where c_s is the source color, α_s is the source alpha, c_d is the destination (background) color

39