Computer Graphics Fundamentals

Dr Samit Bhattacharya

Computer Science and Engineering

IIT Guwahati

Consider a VR Game

What We “See”

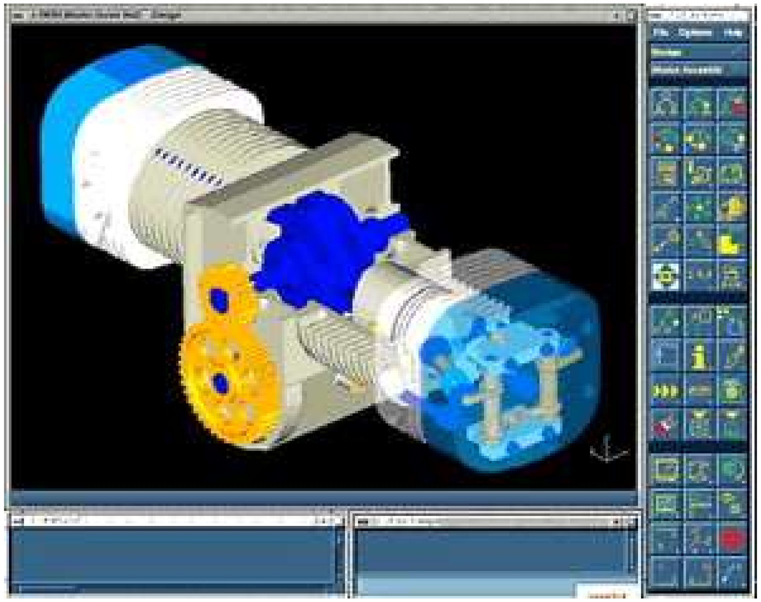
- Images
- Objects
- Texts
- Dynamic change of images
- …

What We "See"

- Broadly, instances of *images* displayed on a screen (text characters can also be considered as images)

What We “See”

- Images are constructed with *objects*, which are <u>basically geometric shapes</u> (characters and icons) with <u>colors assigned</u> to them



What We “See”

- The images or parts can be manipulated (interacted with) by a user
  - Using input devices/methods such as mouse, keyboard, joystick, controllers, smart gloves, gestures and so on

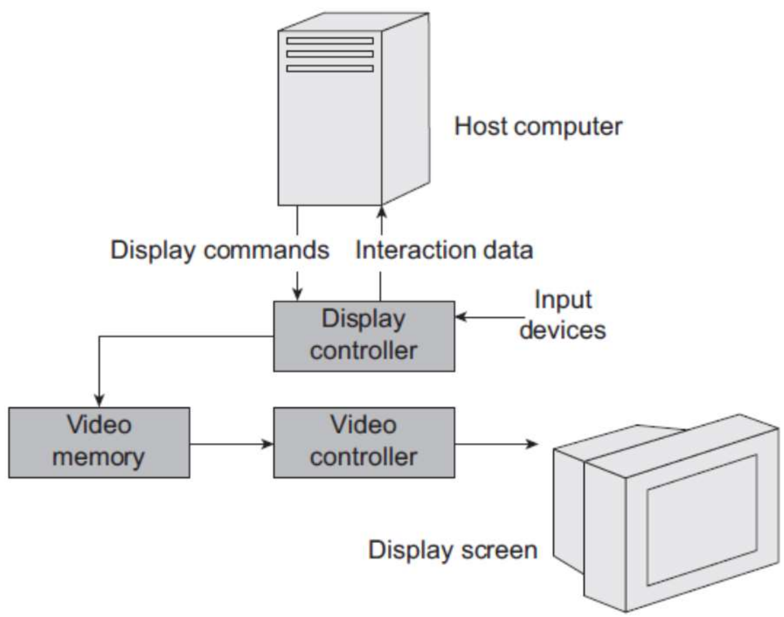
Computer Graphics

- The process of rendering static images or animation (sequence of images) on screen/display in an efficient way

Computer Graphics

- Involves
  - Object representation
  - Object synthesis for scene/image/environment creation
  - Object manipulation
  - Create impression of motion (animation)

Generic CG System Architecture



Display Controller

- Image generation task performed by *display controller*
  - Takes input from CPU (host computer)
  - As well as external *input devices* (mouse, keyboard, etc.)

Display Controller

- Image generation a multi-stage process, involving lots of computation
- Usually carried out by dedicated hardware (*graphics card*)
  - Has own processing unit (GPU or graphics processing unit)

Video Memory

- Display controller generates image in digital format (strings of 0s and 1s)

Video Memory

- Place where it is stored is *video memory*
  - A (dedicated) part of memory hierarchy
  - Typically part of separate graphics unit (the *VRAM* in graphic card)

Video Controller

- Converts digital image to analog voltages
  - Takes stored image as input
  - Analogue voltage drives electro-mechanical arrangements, which ultimately render image on screen

Video Controller

- Display screen contains picture elements or *pixels* (arranged in grid)
- Pixels are *excited* by electrical means → *emit* lights with *specific intensities* → give us *sensation* of colored image on screen

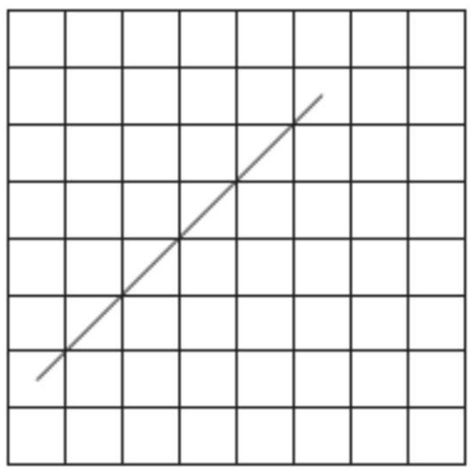
Types of Graphics Devices

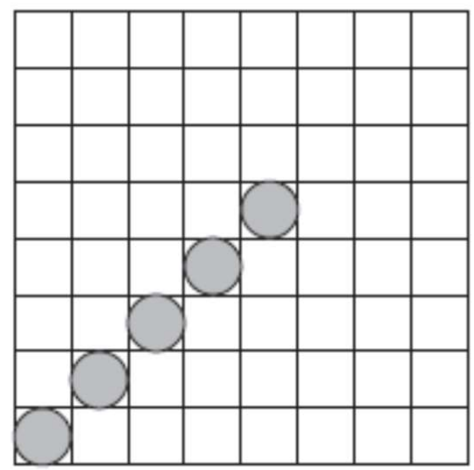
- Broadly two types (based on the method used for excitation of pixels)
  - Vector Scan
  - Raster Scan

Vector Scan Devices

- Also known as *random-scan stroke-writing*, or *calligraphic devices*
- Image *viewed* as composed of continuous geometric primitives such as lines and curves
  - Intuitively what we think of images
- Image rendered by rendering these primitives (only)

Vector Scan Example

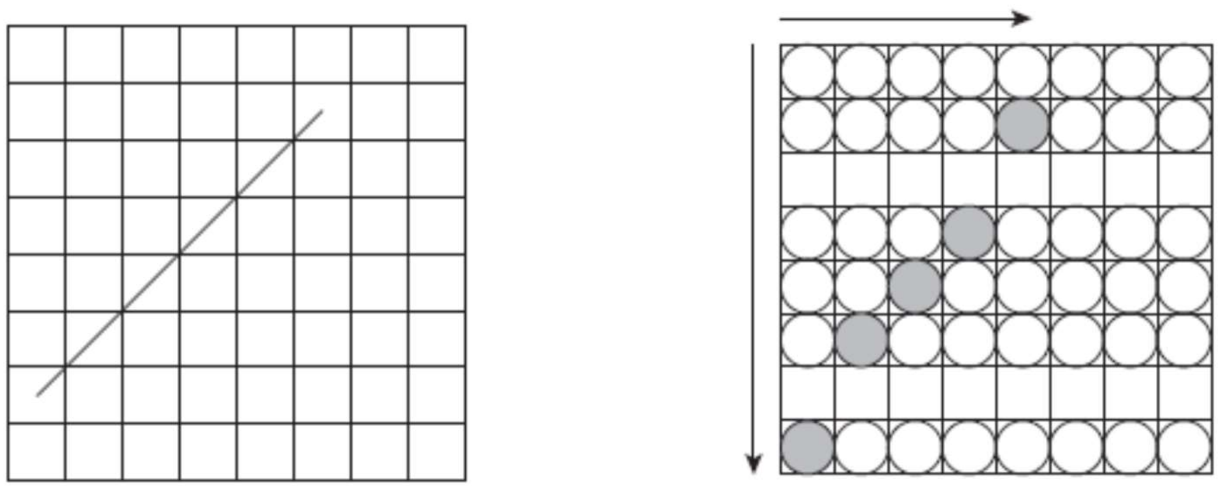




Raster Scan Devices

- Image viewed as represented by WHOLE pixel grid (not only selected pixels representing primitives)
  - To render, ALL pixels are considered
- Achieved by considering pixels in sequence (typically left to right, top to bottom)

Raster Scan Example



Vector & Raster Graphics

- Closely related terms
- Vector graphics – refers to images represented in terms of continuous geometric primitives such as lines and curves
- Raster graphics – refers to images represented in terms of a pixel grid

Black n White Image Generation

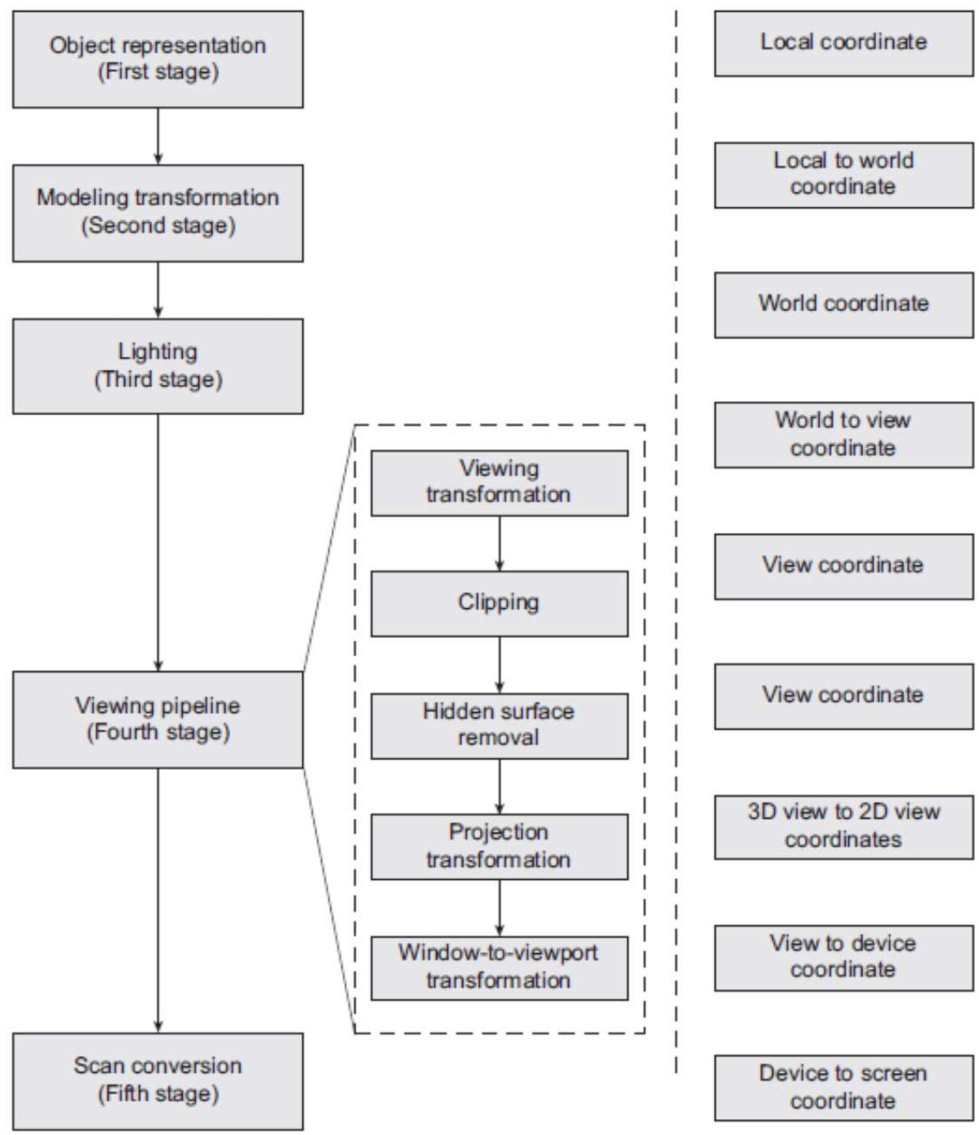
- Each pixel contain one type of element (e.g., a single phosphor dot on a CRT)
  - We can *excite* it to generate different light *intensities* – representing different shades of gray

Color Image Generation

- Each pixel contains more than one type of element (e.g., 3 phosphor dots representing 3 primary colors red, green, blue)
  - When excited, 3 colors combine to produce desired color

Graphics Pipeline

- We talked about color values stored in frame buffer
  - How are these values obtained?
- Display processor *computes* these values in *stages*
- These stages together are known as the *graphics pipeline*



Object representation (First stage)
Modeling transformation (Second stage)
Lighting (Third stage)
Viewing pipeline (Fourth stage)
Scan conversion (Fifth stage)
Viewing transformation
Clipping
Hidden surface removal
Projection transformation
Window-to-viewport transformation
Local coordinate
Local to world coordinate
World coordinate
World to view coordinate
View coordinate
View coordinate
3D view to 2D view coordinates
View to device coordinate
Device to screen coordinate

Object Representation – Broad Types

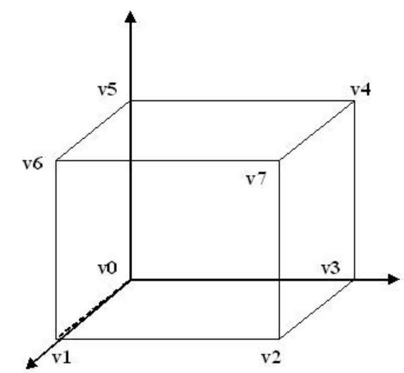
- Boundary (surface) representation
- Space-partitioning
  - Representing region with set of non-overlapping, contiguous solids (usually cubes)

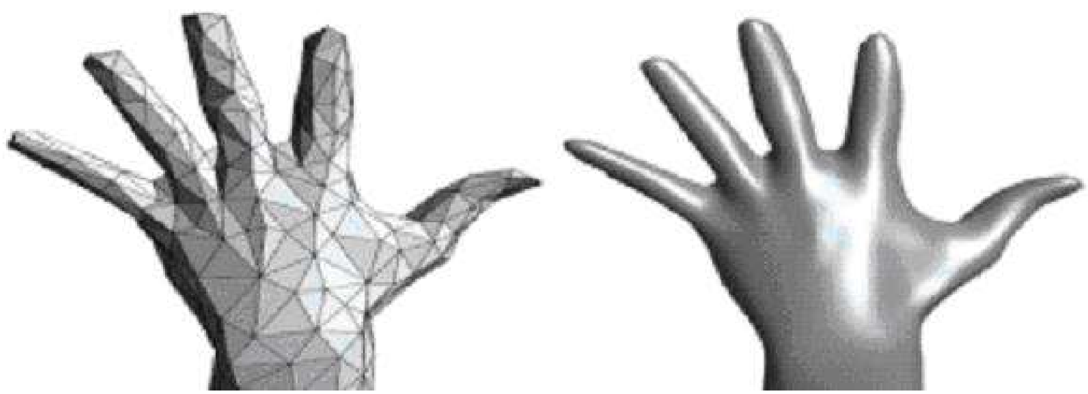
Mesh
(Boundary Representation)

- Connected set of polygons (usually triangles)

Vertex list

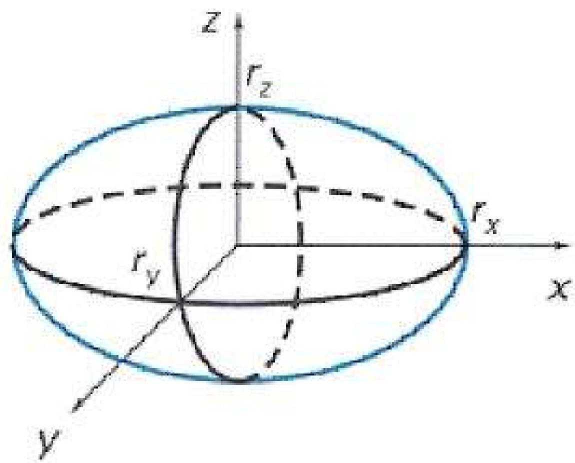
v0	0,0,0	v1v3v5
v1	0,0,1	v6v0v2
v2	1,0,1	v1v3v7
v3	1,0,0	v0v2v4
v4	1,1,0	v3v5v7
v5	0,1,0	v0v4v6
v6	0,1,1	v1v5v7
v7	1,1,1	v2v4v6





Implicit Function

- Surfaces represented by equations
- Ex - Ellipsoid



Implicit form:

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 - 1 = 0$$

Parametric form:

$$x = r_x \cos\phi \cos\theta$$

$$y = r_y \cos\phi \sin\theta$$

$$z = r_z \sin\phi$$

Curve Representation

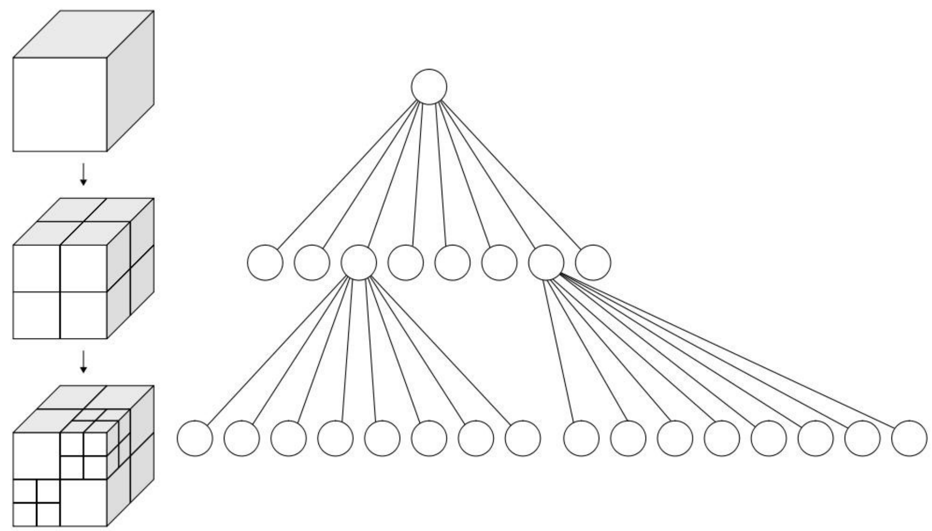
- **Spline representation** (for complex surfaces)

Spline Idea

- Use SEVERAL polynomials
- Complete curve consists of several pieces
  - Each called “blending/basis function”
- All pieces are of low order
  - Third order most common
- Pieces join smoothly

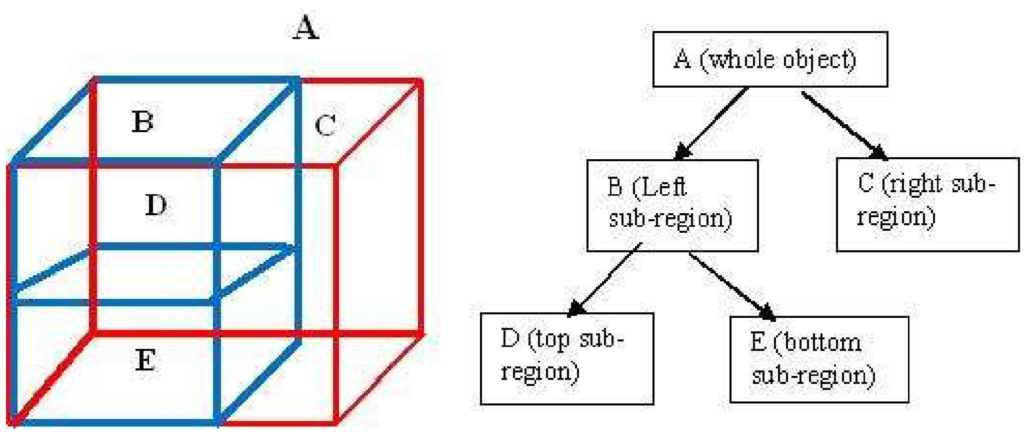
Voxels (Space Partitioning)

- Partition space into uniform grid
  - Grid cells are called voxels (like pixels)



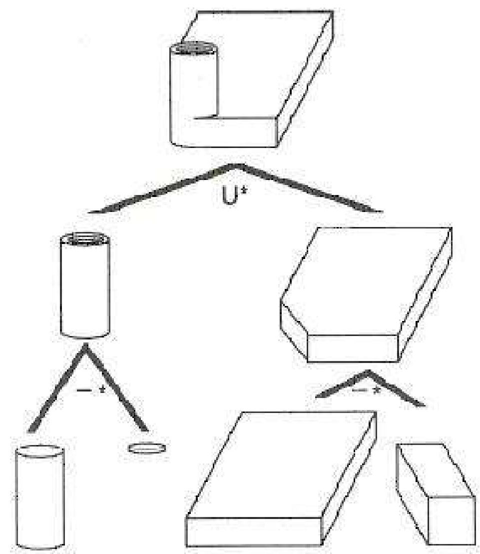
Binary Space Partitions (BSPs)

- Recursive partition of space by planes
  - Mark leaf cells as inside or outside object



Constructive Solid Geometry (CSG)

- Represent solid object as hierarchy of Boolean operations
  - Union
  - Intersection
  - Difference

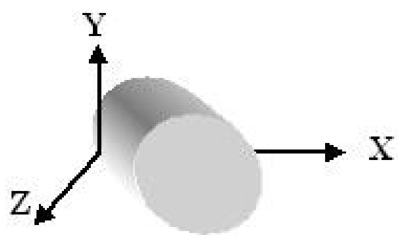


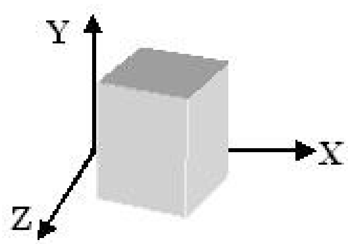
Other Representations

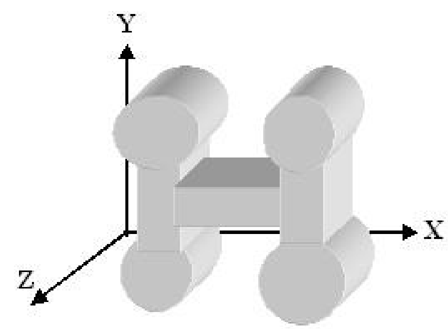
- **Sweep representation -** geometric shape + trajectory defines object
- **Fractals -** procedural representation technique
- **Particle system** –physically-based modeling technique
- **Skeletal models** – hierarchy of ‘bones’ (inner layer) covered with ‘skin’ (outer layer)
- **Scene graphs** – a graph-like data structure for objects in a scene
- Application specific

Modeling Transformation

- Objects of a scene are individually represented in their own reference frames (object/local coordinate systems)
- Through modeling transformations, objects are combined into a world coordinate scene

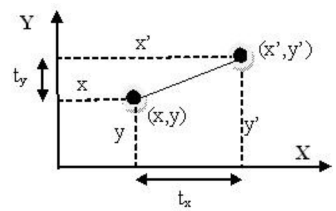




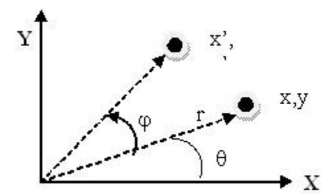


Basic Transformations (2D)

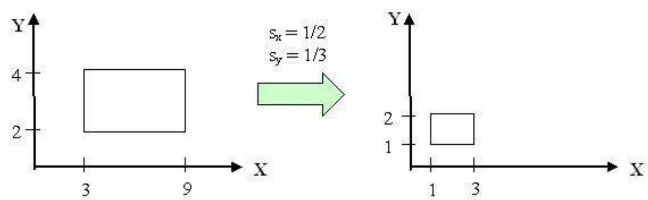
- Translation



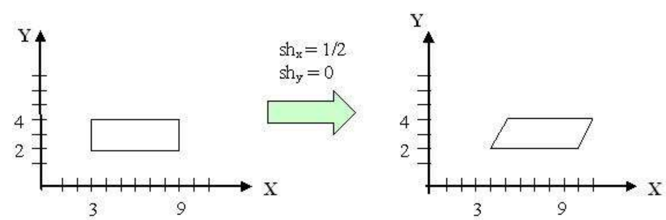
- Rotation



- Scale



- Shear



Matrix Representation

- Represent 2D transformation by a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- Multiply matrix by column vector $\Leftrightarrow$ apply transformation to point

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned}$$

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices (homogeneous coordinate system)

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \boldsymbol{t}_x \\ 0 & 1 & \boldsymbol{t}_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

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

Rotate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Basic 3D Transformations

Identity

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

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ \boldsymbol{z}' \\ \boldsymbol{w} \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 & 0 \\ 0 & 0 & \boldsymbol{s}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{w} \end{bmatrix}$$

Scale

Translation

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ \boldsymbol{z}' \\ \boldsymbol{w} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \boldsymbol{t}_x \\ 0 & 1 & 0 & \boldsymbol{t}_y \\ 0 & 0 & 1 & \boldsymbol{t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{w} \end{bmatrix}$$

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

Mirror about Y/Z plane

Basic 3D 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:

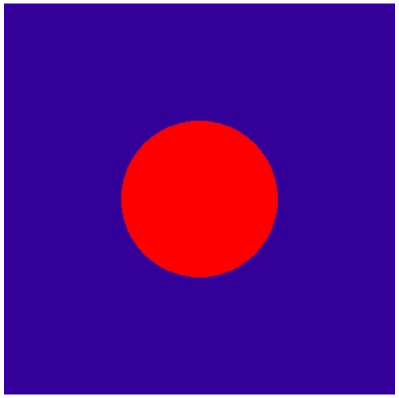
$$\begin{bmatrix} \boldsymbol{x'} \\ \boldsymbol{y'} \\ \boldsymbol{z'} \\ \boldsymbol{w} \end{bmatrix} = \begin{bmatrix} \cos\Theta & 0 & \sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\Theta & 0 & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{w} \end{bmatrix}$$

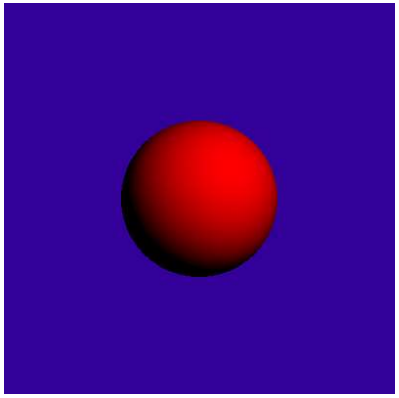
Rotate around X axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & -\sin\Theta & 0 \\ 0 & \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Illumination (Adding Color) – Why?

- If we don’t have lighting effects nothing looks three dimensional!





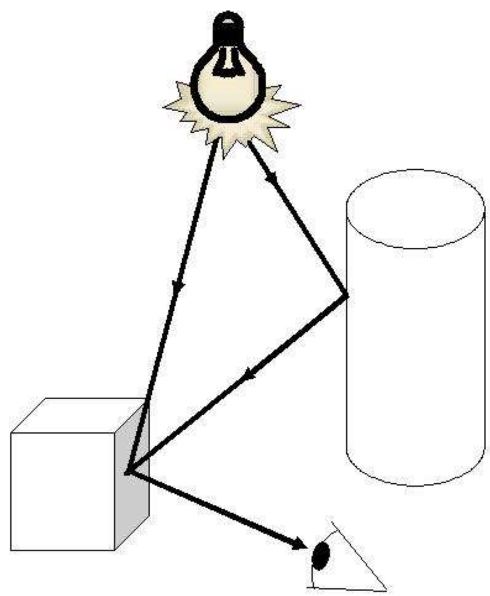
Some Definitions

- **Illumination**: transport of energy from light source to surfaces & points
  - Note: includes *direct* and *indirect illumination*

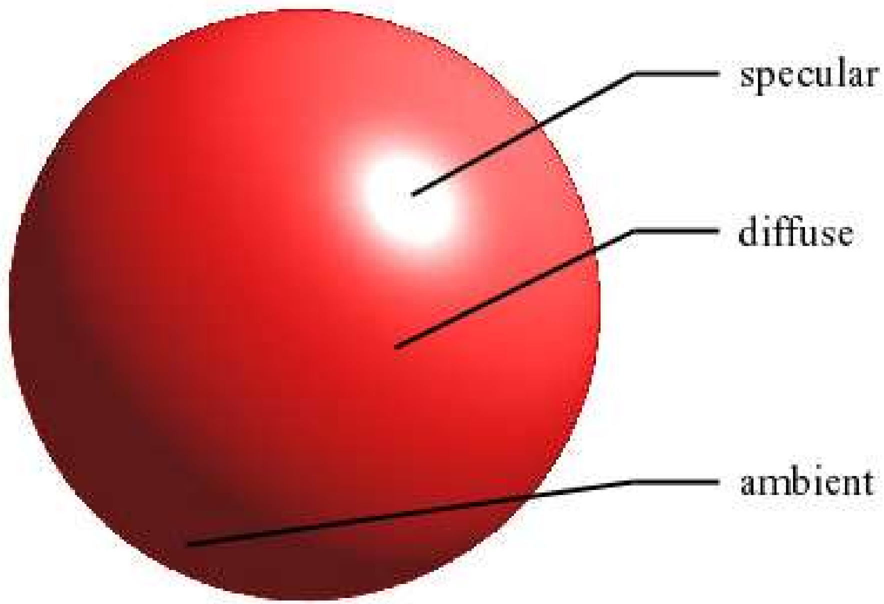
- **Lighting**: process of computing the luminous intensity (i.e., outgoing light) at a particular 3-D point, usually on a surface

- **Shading/surface rendering**: process of assigning colors to pixels

Perception of Color



Reflection Types



Basic Illumination Model

- Important components
  - Ambient light
  - Diffuse reflection
  - Specular reflection

$$I_p = I_{amb} + I_{diff} + I_{spec}$$

Diffuse Reflection – Ambient Light

- For background lighting effects, assume every surface fully illuminated by ambient light I_a

- Therefore, ambient contribution to diffuse reflection is

$$I_{ambdiff} = k_d I_a$$

Diffuse Reflection

- Amount of incident light on a surface (following Lambert's law)

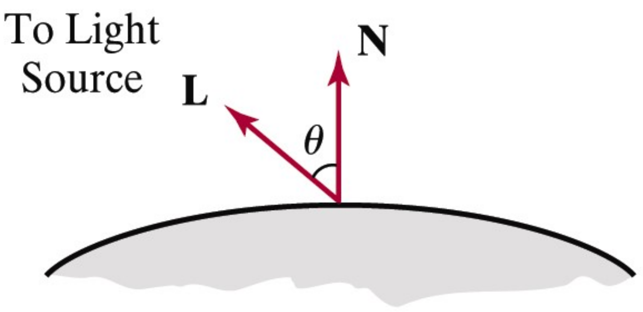
$$I_s \cos\theta$$

- Diffuse reflections component is

$$I_{diff} = k_d I_s \cos\theta$$

Diffuse Reflection

- N = surface normal, L = unit direction vector to the light source

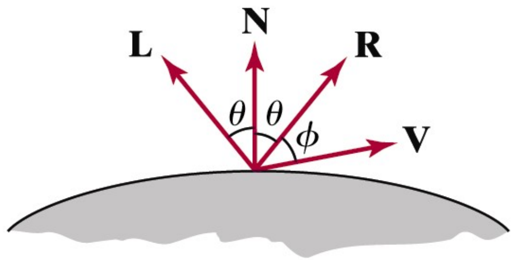


- Then, $N \cdot L = \cos\theta$

- Thus,

$$I_{diff} = \begin{cases} k_d I_s (N \cdot L) & \text{if } N \cdot L > 0 \\ 0 & \text{if } N \cdot L \leq 0 \end{cases}$$

Specular Reflection



- Specular reflection intensity

$$I_{spec} = k_s I_s \cos^{n_s} \phi$$

$$I_{spec} = \begin{cases} k_s I_s (V \cdot R)^{n_s} & \text{if } V \cdot R > 0 \text{ and } N \cdot L > 0 \\ 0.0 & \text{if } V \cdot R < 0 \text{ or } N \cdot L \leq 0 \end{cases}$$

- *R* can be represented as *(2N.L)N-L*

Things We Ignored!

- Intensity attenuation (spatial, angular)
- Multiple light source
- Surface transparency

Applying Illumination

- Computing color
  - Fairly expensive calculation
- Shading/surface rendering methods

Flat Surface Rendering

- Simplest method for rendering a polygon surface - same color assigned to all surface positions

- Illumination at a single point on the surface calculated and used for entire surface

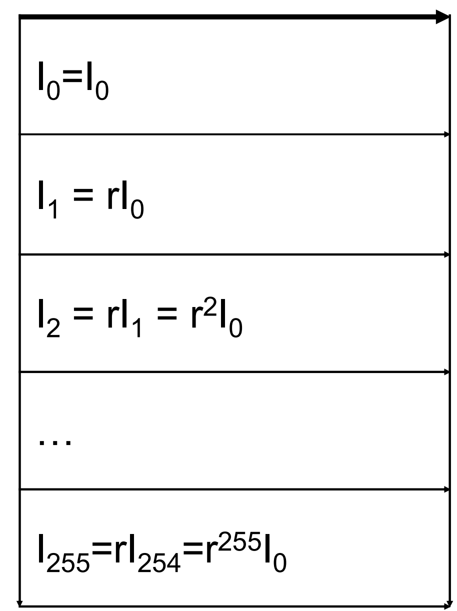
- Extremely fast, but can be unrealistic

Intensity Representation

- Illumination model gives intensity as any value in the range of 0.0 to 1.0
- A graphics system can display only a limited set of intensity values
- Calculated intensity must be converted to one of the allowable system values (without affecting perception)

Representing Intensities

- $I_1/I_0=I_2/I_1=\ldots=I_n/I_{n-1} = r$
- $I_k = r^k I_0$, $k > 0$
- Ex: B/W monitor with 8 bits/pixel
  - $n = 255$
  - $r = 1.0182$ (typical)
  - $I_0 = 0.01$ (say)
  - Ints = 0.0100. 0.0102, 0.0104…1…
  - Assign 256 bit patterns to the 256 intensities



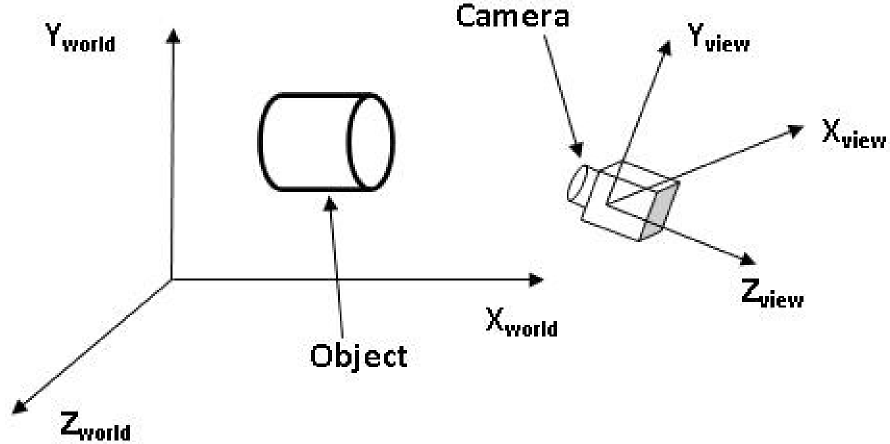
Intensity – IM to Device

- Let I = intensity value calculated by an illumination model (IM)

- Calculate nearest intensity level I_k supported by the device (from a table of pre-computed intensity values) – assign bit patterns to those levels

3D Viewing

- Just like taking a photograph!
- World coordinates to viewing coordinates:
  viewing transformations



Camera Parameters

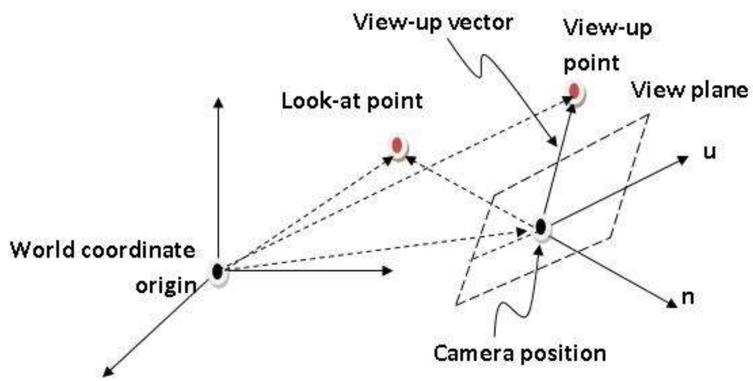
- Important camera parameters to specify
  - Camera (eye) position in world coordinate system
    - Also called *viewpoint/viewing position*
  - Center of interest
    - Also called *look-at point*
  - Orientation (which way is up?) View-up vector

View Coordinate Frame

- Known: eye position, center of interest, view-up vector
- To find out: new origin and three basis vectors

View Coordinate Frame

- Put it all together



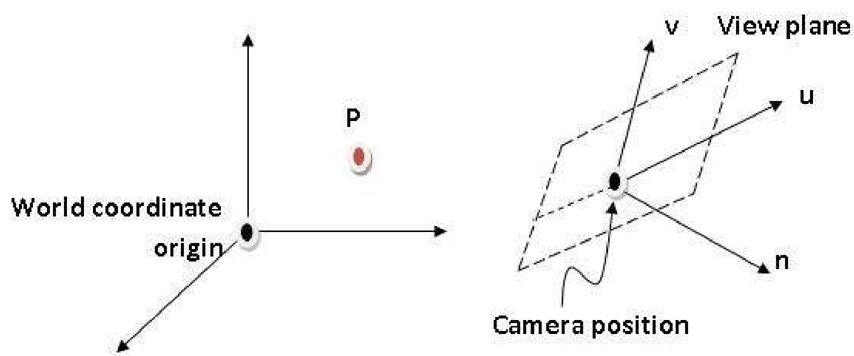
Eye space **origin: (ex , ey, ez)**

Basis vectors:

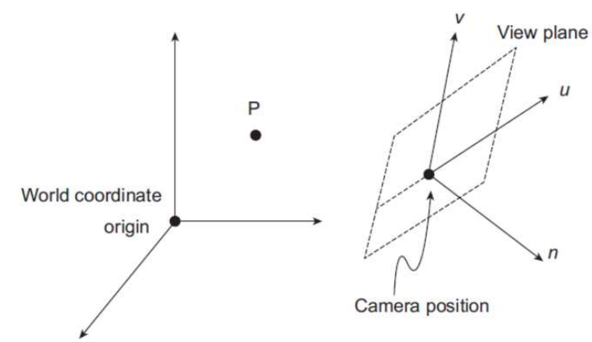
n = (eye – COI) / | eye – COI|
u = (V_up x **n**) / | V_up x **n** |
v = **n** x **u**

WC → VC Transformation

- Transform object description from WC to VC
- Transformation matrix (Mw2e); P' = Mw2v . P



WC → VC Transformation



- M_{w2v} = (Rotation matrix).(Translation matrix)

$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T = \begin{bmatrix} 1 & 0 & 0 & -o_{vx} \\ 1 & 0 & 0 & -o_{vy} \\ 1 & 0 & 0 & -o_{vz} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Transformation

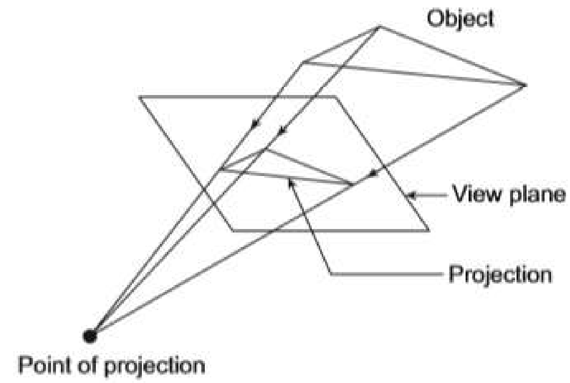
- Once WC→VC transformation is done, the 3D objects are projected on the 2D view plane

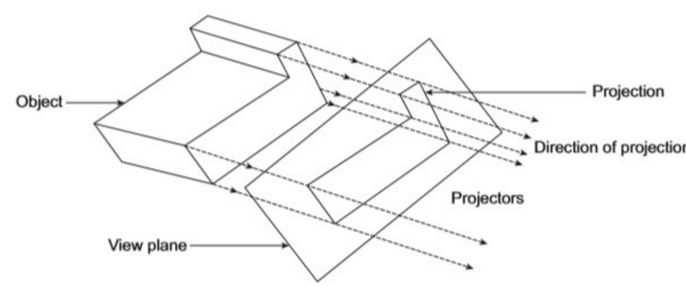
Projection

- Map objects from 3D space to 2D screen
  - Defined by straight lines - *projectors*

Planar Geometric Projections

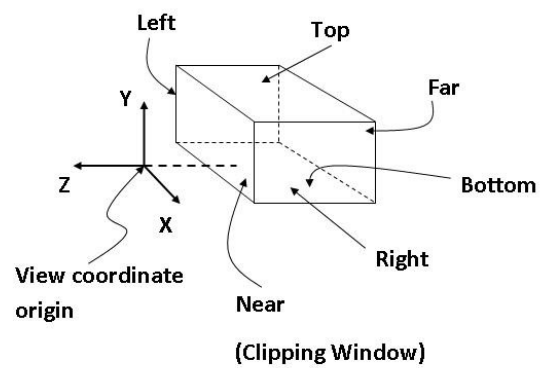
- Projectors are lines that either
  - Converge at a center of projection (**perspective projection**)
  - Are parallel (**parallel projection**) – center of projection at infinity

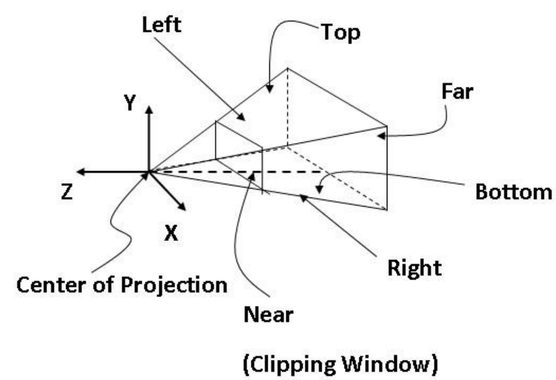




Projection Transformation

- Important things to control (to define view volume)
  - **Parallel projection** – view volume is rectangular parallelepiped
  - **Perspective projection** – view volume is a frustrum

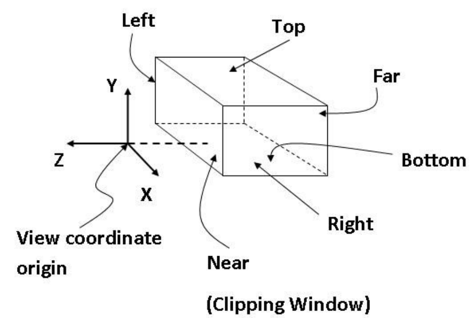




Parallel Projection

- Matrix form (assuming view plane at a distance d from origin, along the –z direction)

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

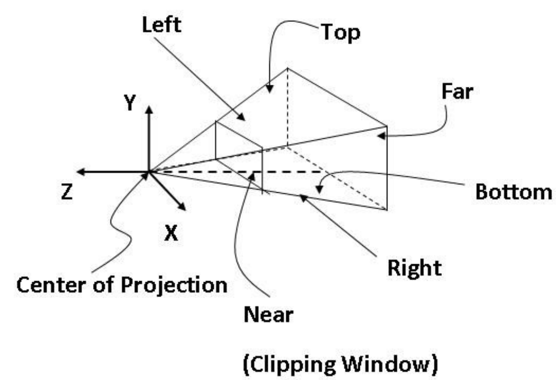


- Note that this is in homogeneous coordinate
  - x' = the actual projected point = x''/w etc…

Perspective Projection

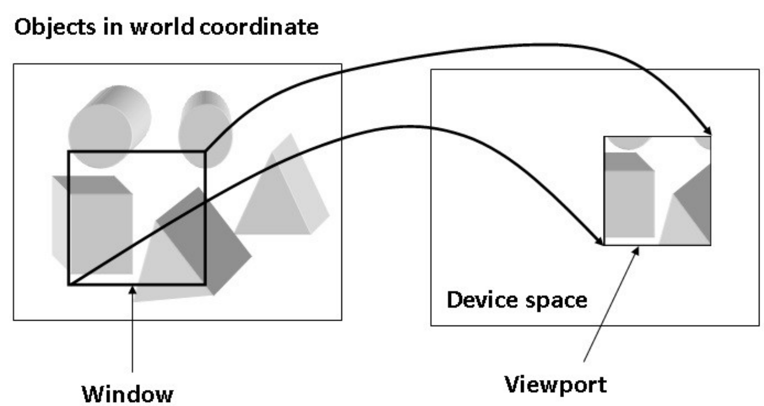
- Matrix form (in homogeneous coordinate system)

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Viewport Transformation

- Transform from projection coordinates (normalized clipping window) to device coordinates



Window vs Viewport

- Window
  - World-coordinate area selected for display
  - What is to be viewed
- Viewport
  - Area on the display device to which a window is mapped
  - Where it is to be displayed

Viewport Transformations

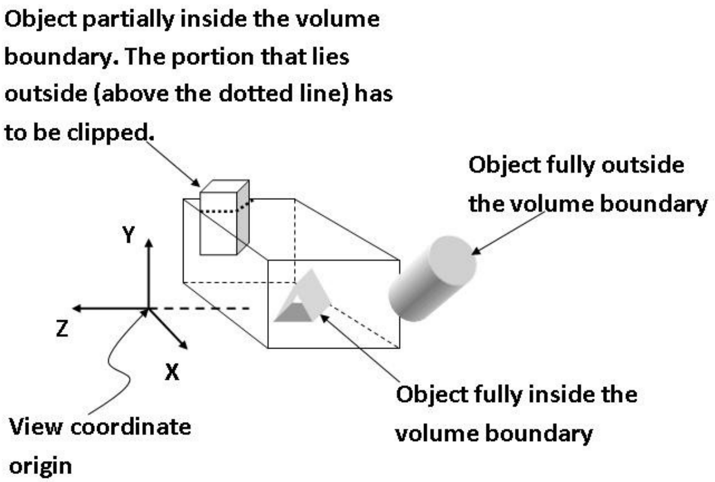
- Transformation matrix,

$$M_{WV} = \begin{bmatrix} sx & o & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix}$$

- sx ≠ sy, transformed object will be scaled (up/down)

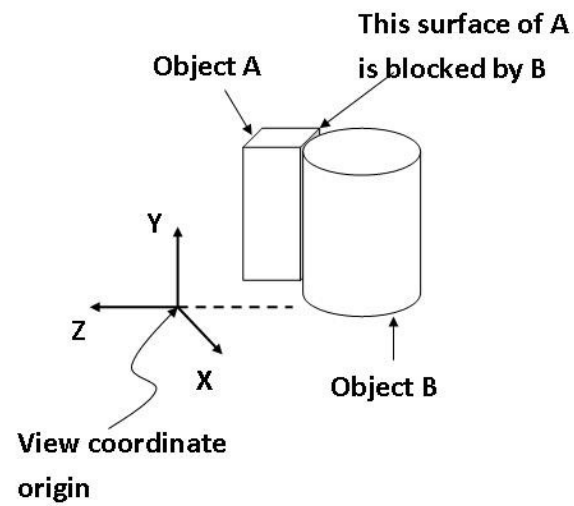
Clip Objects

- Objects that are partially within the viewing volume need to be clipped



Hidden Surface Removal

We must determine what is visible within a scene from a chosen viewing position



Note

- Both clipping and HSR done by application of corresponding algorithms
  - Many algorithms available for both

Rendering

- Pipeline stages covered so far assumed continuous space
  - Methods considered points without any constraint on the coordinates – can be any real number
- To draw something on the screen, we need to consider pixel grid
  - Discrete space

Rendering

- Need to map from continuous to discreet space
- Mapping process called rendering
  - Also called *scan conversion/rasterization*

Line Scan Conversion

- line segment - defined by coordinate positions of line end-points

- What happens when we try to draw this on a pixel based display?
  - How to choose the correct pixels

Other Algorithms

- Circle rendering
- Curve rendering
- Surface rendering
- Character rendering
- Anti-aliasing (to make the rendering smooth)

Conclusion

- Virtual object/scene rendering is computation intensive
  - Previous discussion meant to give you an idea
  - We haven’t discussed interactive manipulation of objects (involves even more computations)
  - Plus, animation (moving frames/images) – yet more computation

Good News!

- You may never require to learn it
  - You can create objects/scenes with UIs (e.g. Unity UI, Maya …)
    - Such UIs come equipped to help you create and manipulate objects
  - You can use graphics libraries to create s/w of your own (e.g. OpenGL)

It always helps to be better informed

Advantage of Knowledge

- Will be able to better appreciate requirements (why XR systems require high end machines)
- Utility of cloud-based support (if you don't have high end machines)
- Problem with your systems (why it slows down/hangs) - mismatch between availability of resources and your requirements (high-quality photo-realistic rendering)
- Will be able to build your own s/w!