Virtual Reality and Augmented Reality

August 17, 2023

What is Computer Graphics?

Computer Graphics

The term Computer Graphics roughly refers to the field of study that deals with the display mechanism (Hardware and Software) of a Computer.

Computer Graphics

In the early days, for most of us, computer meant what we got to see on the "MONITOR" (of a Desktop/Laptop)

Computer Graphics

- For younger generation, PC/Laptop no longer the only 'COMPUTER' - replaced by a plethora of devices.
- These devices come in various shapes and sizes with varying degrees of functionality.
- For example: Smartphones, tables (or tabs), wearable computers (smart watch, smart glass), even televisions (smart TVs).
- Made possible with rapid technological change, including display technology: CRT replaced by LCD, plasma panel, light-emitting diode (LED), organic light-emitting diode (OLED), thin-film transistor (TFT).

Computer Graphics

Regardless of the state-of-the-art technology in computing, the idea of a computer is shaped primarily by what we got to see on the display.

Computer Graphics

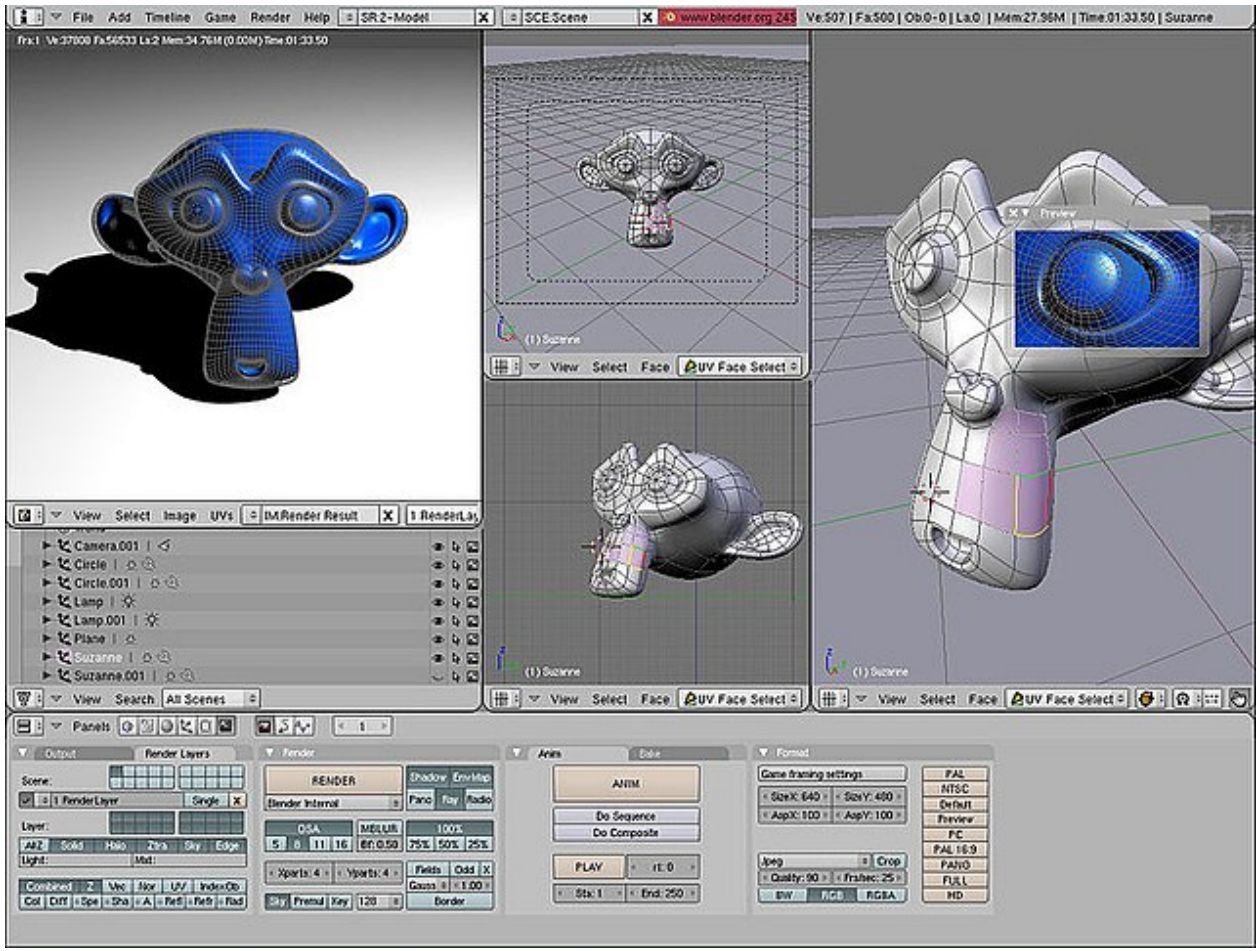
It is important to know components of a computing system that give rise to this perception - the display hardware and the associated software and algorithms.

What is Computer Graphics

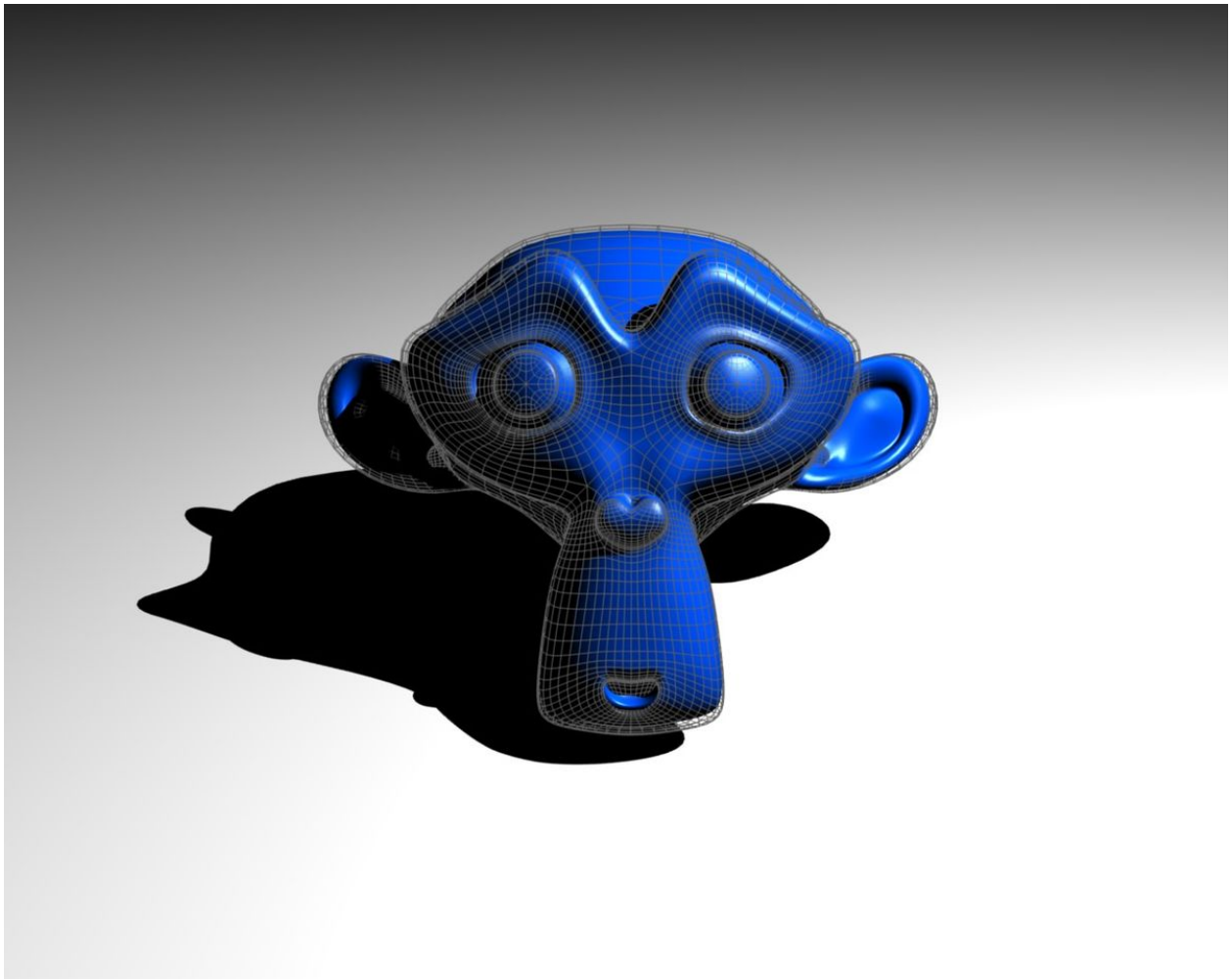
Anything to do with visual representations on a computer

- Modelling
- Rendering
- Also animation - dynamics of moving objects

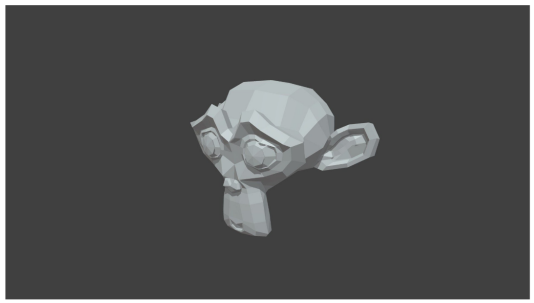
Modelling

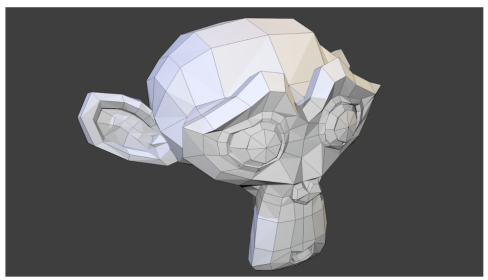

File Add Timeline Game Render Help
SR:2-Model
SCE:Scene
www.blender.org 245
Ve:507 | Fa:500 | Ob:0-0 | La:0 | Mem:27.96M | Time:01:33.50 | Suzanne
View Select Image UVs
IM:Render Result
Camera.001
Circle
Circle.001
Lamp
Lamp.001
Plane
Suzanne
Suzanne.001
View Search All Scenes
View Select Face UV Face Select
Panels
Output
Render Layers
RENDER
ANIM
PLAY
Do Sequence
Do Composite
Format
PAL
NTSC
Default
Preview
PC
PAL 16:9
PAN0
FULL
HD

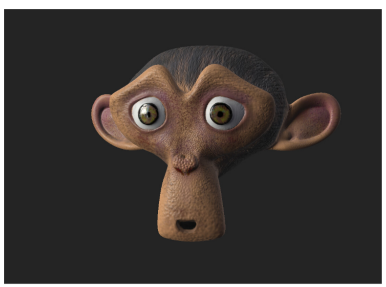
Rendering



Animation





Computer Graphics

What We Do With Computers?

What We Do With Computers?

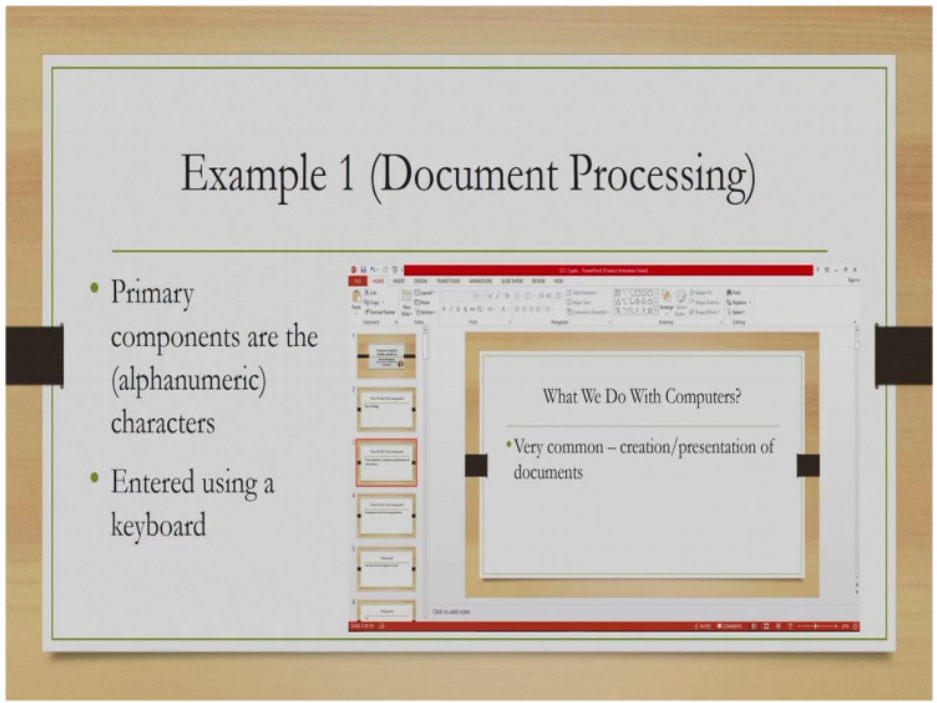
Lot of things!!!

What We Do With Computers?

Example 1 (Document Processing)

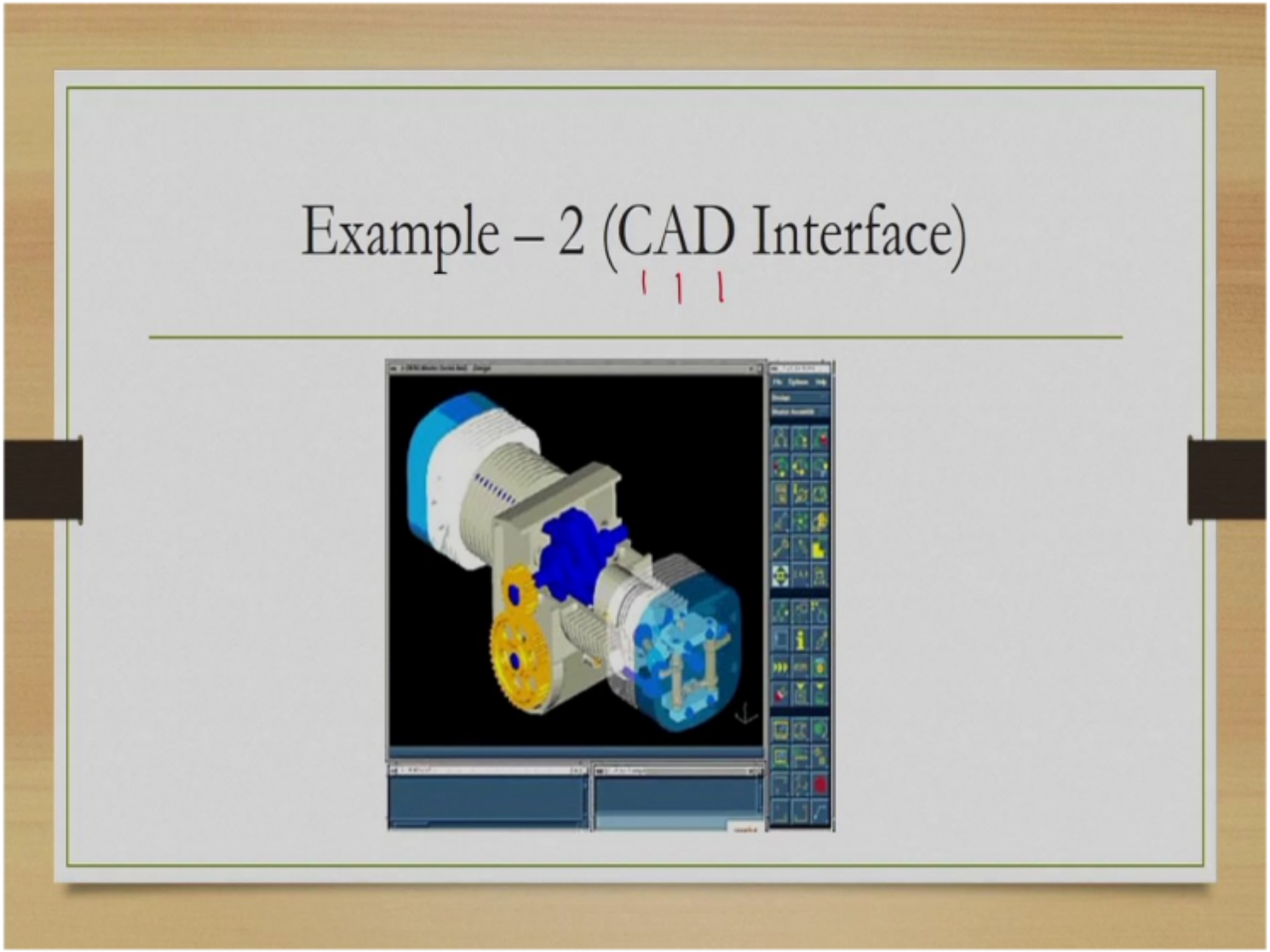
- Creation/Presentation of Documents

What We Do With Computers?

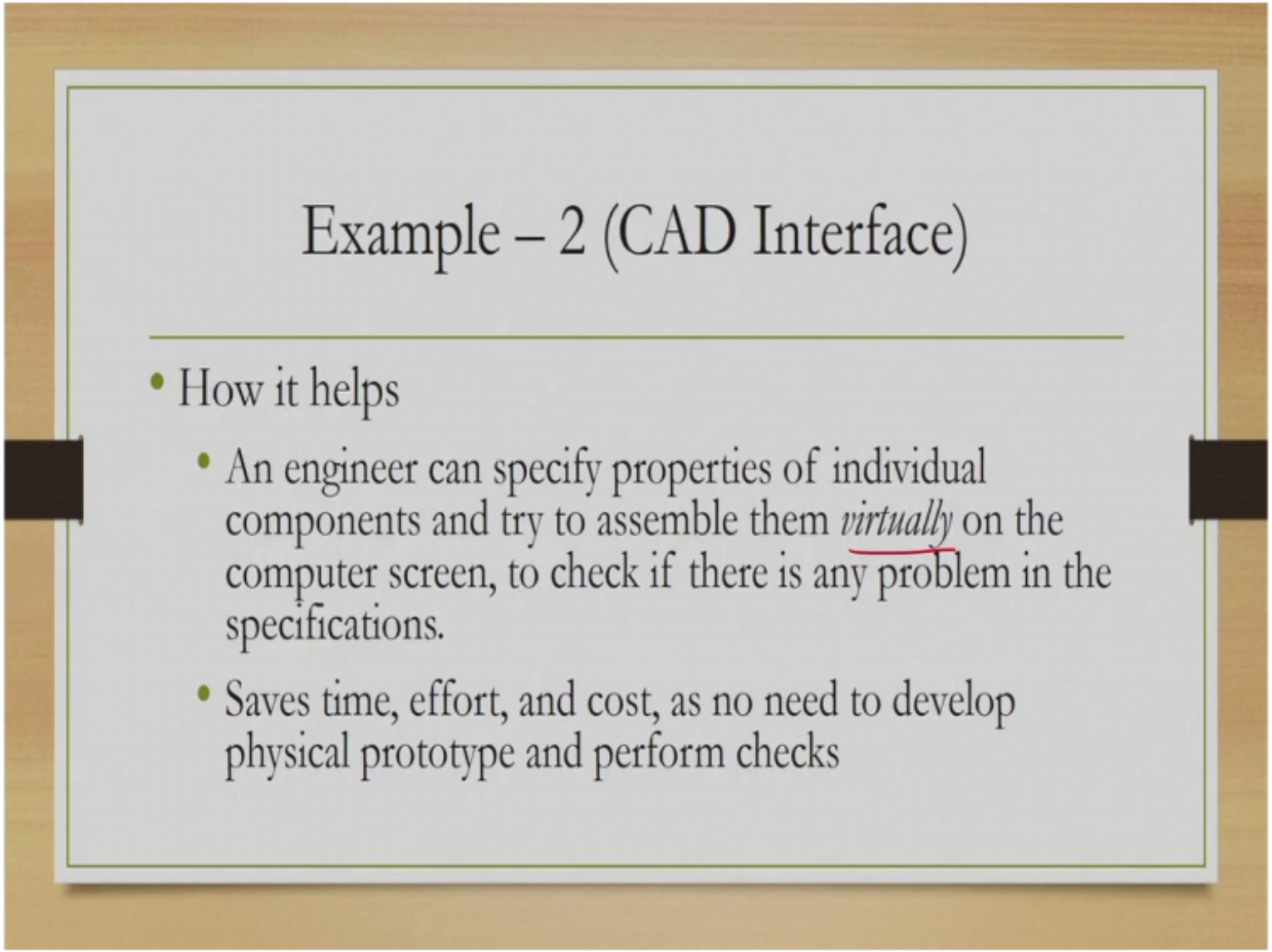


- There are other equally important components
- Such as, the menu option that we see here on the top side of the screen
- We also have the preview slides on the left part which is another important component

What We Do With Computers?



What We Do With Computers?

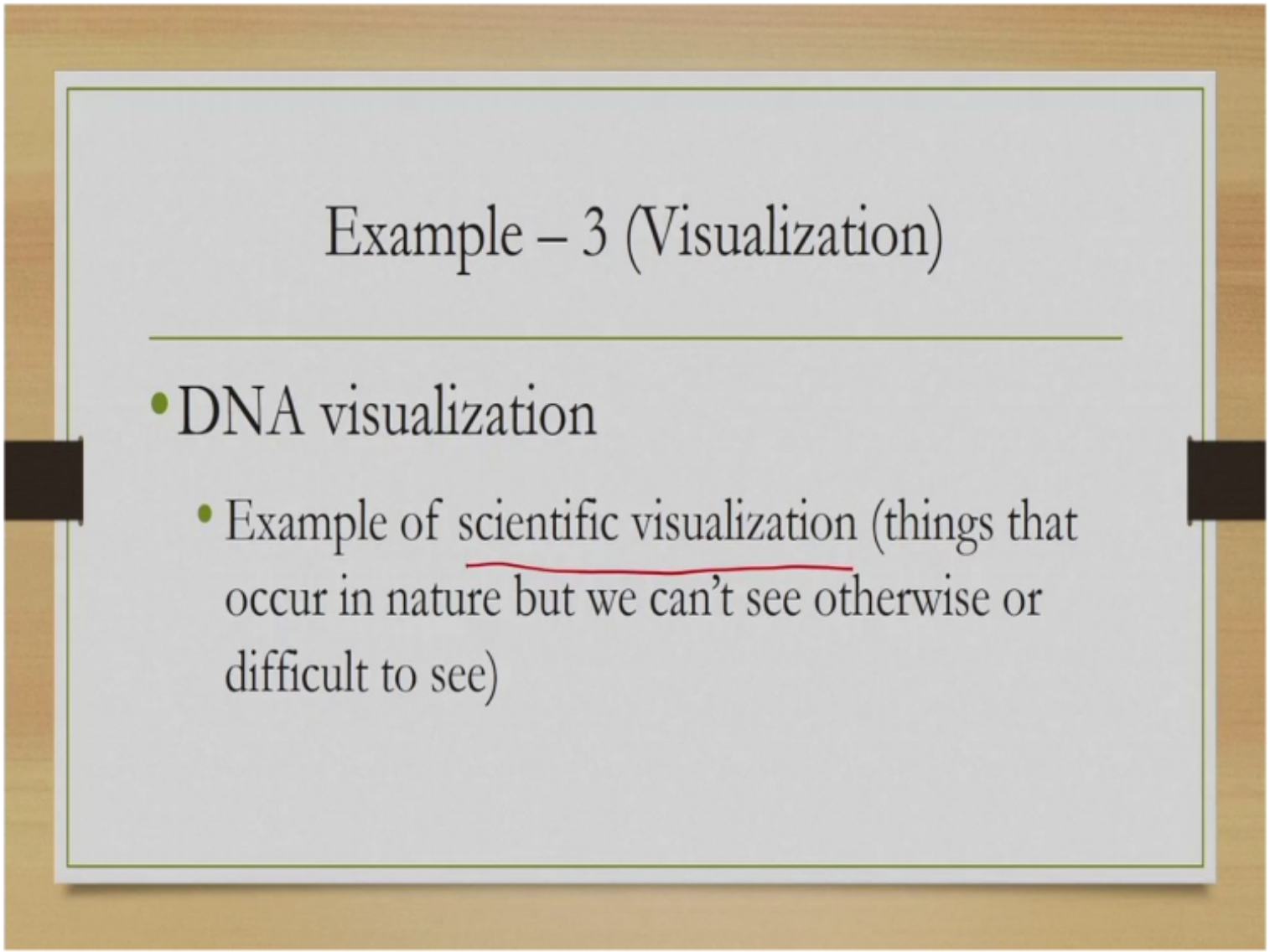


What We Do With Computers?

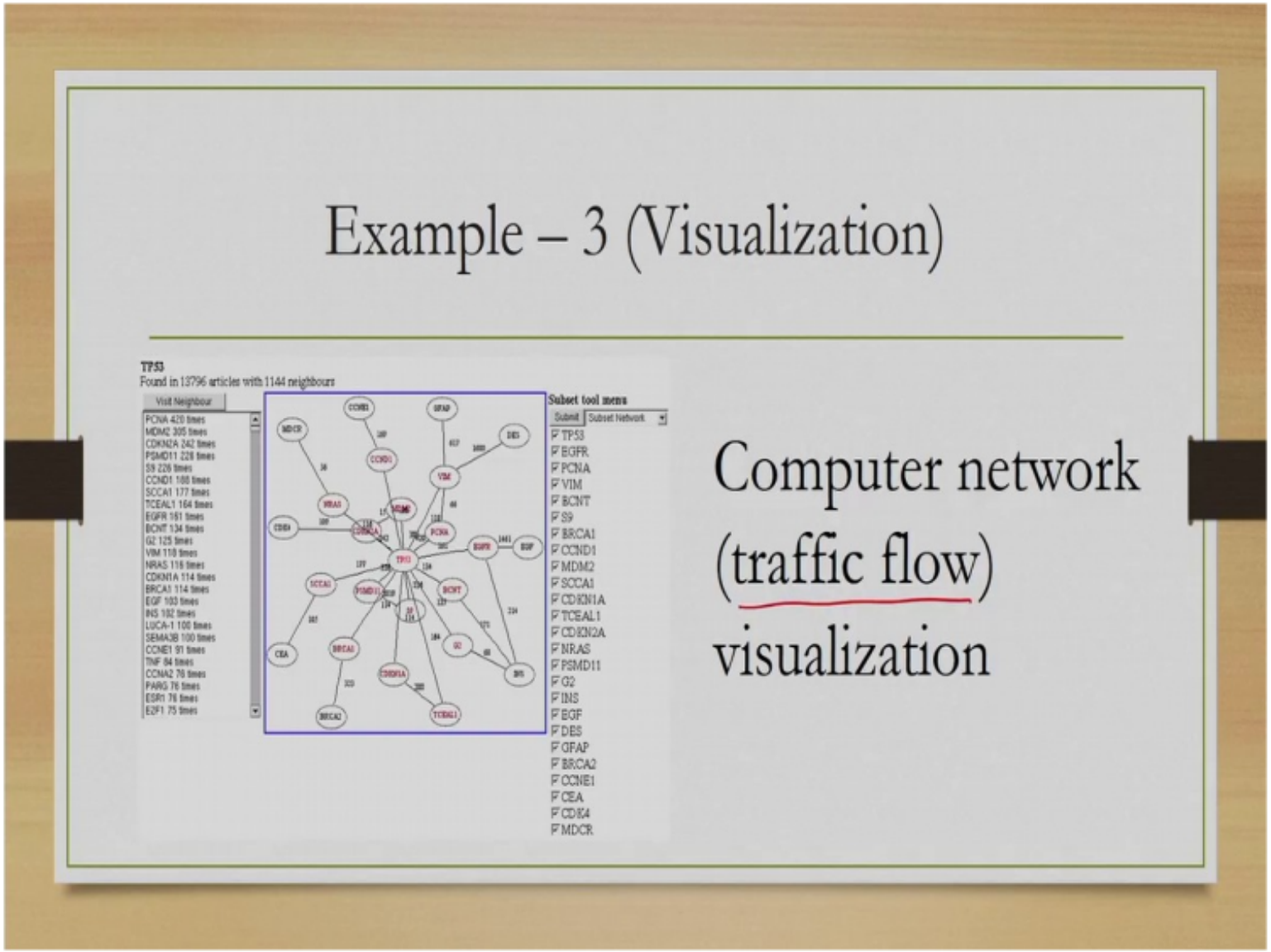
Example – 3 (Visualization)

DNA visualization

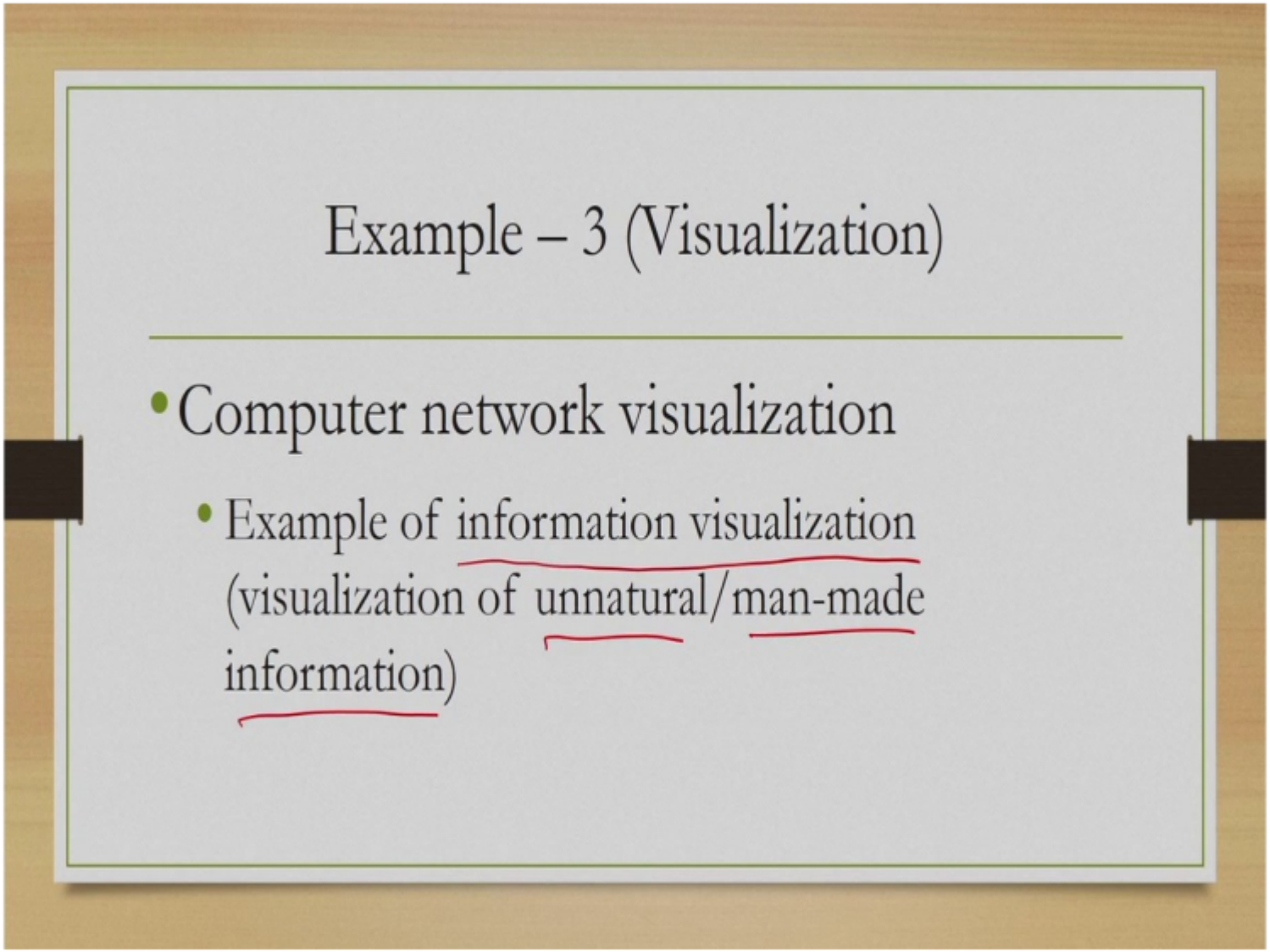
What We Do With Computers?



What We Do With Computers?



What We Do With Computers?



Computer Graphics

- Each an example of usage of Computer graphics
- Each and everything that we get to see around us involving computers are basically applications of computer graphics and it is definitely not possible to least all those applications.
- Computer Graphics used in Mobile phones, information kiosks at popular spots such as airports, ATMs, large displays at open air music concerts, air traffic control panels even movie screens in the theatres all these are some kinds of display and whatever is being shown on this displays are mostly applications of computer graphics.

Computer Graphics

What is common in all these applications?

- Instances of images that are displayed
- Images are constructed with objects, which are basically geometric shapes (characters and icons in example 1 and 2) with colors assigned to them.
- When we create edit or view a document we are dealing with alphanumeric characters and each of these characters is an object.

Computer Graphics?

How can a computer do all these stuff?

- We know computers understand only binary language of 0s and 1s.
- Letters, numbers, symbols, or characters are definitely not strings of Os or 1s!

Motivation

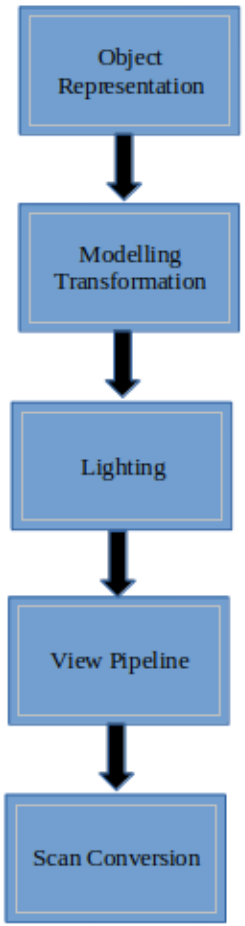
- How we can represent such objects in a language understood by computers so that can be processed by computer?
- How can we map from the computer's language to something that we can perceive (with physical properties such as shape, size and color)
- In other words, how we can create or represent synthesize and render images on a computer display?

Computer Graphics

In computer graphics, we seek answer to these questions.

- In summary, we can say "Computer Graphics" is the process of rendering static images or animation (sequence of images) on computer screens in an efficient way.

Computer Graphics Pipeline



Object Representation

- Object representation techniques allow us to represent objects individually (local/object coordinate system).

Object Representation

- At the item of defining the objects, their shape, size, and position are not important.

Object Representation

- When we are going to compose a scene, the objects need to be assembled together in a different setting so-called scene/world coordinate system.

Object Representation

- When individual objects are assembled in a scene, their shapes, sizes, and positions become very important.

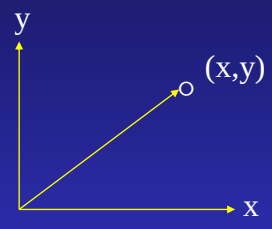
References

- Introduction to Computer Graphics by David J. Eck
  https://math.hws.edu/eck/cs424/downloads/graphicsbook-linked.pdf
- Computer Graphics by Dr. Samit Bhattacharya (Special Thanks to Dr. Samit Bhattacharya)
  https://india.oup.com/product/computer-graphics-9780198096191
- OpenGL https://en.wikipedia.org/wiki/OpenGL
- OpenGL Programming Guide
  https://www.glprogramming.com/red/chapter01.html

2 - D Transformation

1

Representation of Points



A point (x, y) is represented in 2 – D as a vector : $\begin{bmatrix} x \\ y \end{bmatrix}$

2

Transformation of Points

The initial coordinates x and y are transformed by the matrix **T**, to x^* and y^*.

$$\mathbf{TX} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

3

Identity

When the transformation matrix, **T**, is the identity matrix the re is no change in the points, x, y.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

4

Scaling

x – scaling

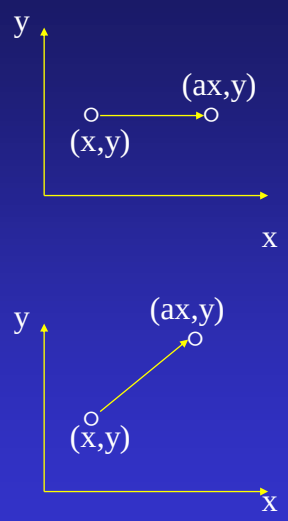
$$\begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} ax \\ y \end{bmatrix}$$

y – scaling

$$\begin{bmatrix} 1 & 0 \\ 0 & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} x \\ dy \end{bmatrix}$$

combined x & y scaling

$$\begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} ax \\ dy \end{bmatrix}$$



5

Reflection

reflection about x – axis

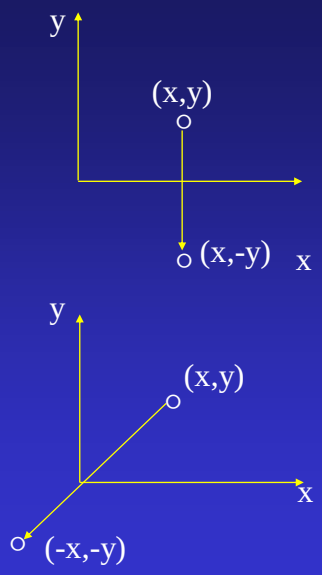
$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} x \\ -y \end{bmatrix}$$

reflection about y – axis

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}$$

reflection through origin

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} -x \\ -y \end{bmatrix}$$



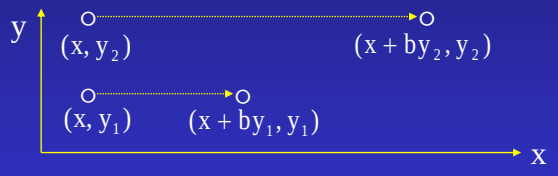
6

Shear

So far all transformations have been on the main diagonal.

Shear, x – coordinate

$$\begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} x + by \\ y \end{bmatrix}$$



Shear, y – coordinate

$$\begin{bmatrix} 1 & 0 \\ c & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} x \\ cx + y \end{bmatrix}$$

7

Origin

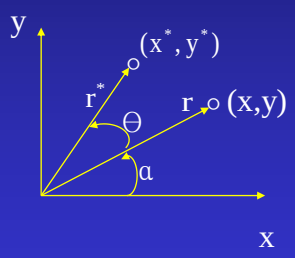
The origin is invariant to these types of trnasformations.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

8

Rotation – 1

Rotation about the origin:



in polar coordinate s :

$x = r\cos\alpha \qquad y = r\sin\alpha$

$x^* = r\cos(\alpha+\theta) \qquad y^* = r\sin(\alpha+\theta)$

where : length of $\vec{r} = \|r\| = r$

Rotation – 2

using some trigonometry

$$x^* = r\cos\alpha\cos\theta - r\sin\alpha\sin\theta$$
$$= (r\cos\alpha)\cos\theta - (r\sin\alpha)\sin\theta$$
$$= x\cos\theta - y\sin\theta$$
$$= (\cos\theta)x - (\sin\theta)y$$

$$y^* = r\cos\alpha\sin\theta + r\sin\alpha\cos\theta$$
$$= (r\cos\alpha)\sin\theta + (r\sin\alpha)\cos\theta$$
$$= (\sin\theta)x + (\cos\theta)y$$

rotation transformation

$$\begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x^* \\ y^* \end{bmatrix}$$

Sequential Transformations

Since matrix multiplication is non-commutative, i.e. AB≠BA (in general) ⇨ the order of transformation is important

i.e $$\mathbf{T}_b\,\mathbf{T}_a\begin{bmatrix} x \\ y \end{bmatrix} \neq \mathbf{T}_a\,\mathbf{T}_b\begin{bmatrix} x \\ y \end{bmatrix}$$

11

Example

transformations – rotation, $\theta = 135^\circ$

reflection thru x – axis

$$\text{let}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

rotate then reflect

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ 0 \end{bmatrix}$$

reflect then rotate

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix}$$

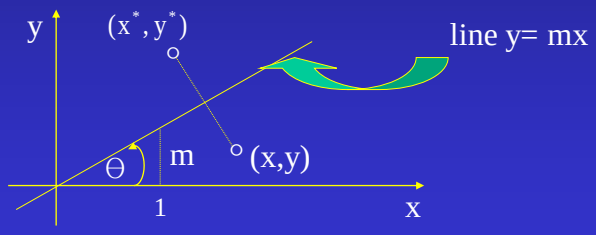
12

Combined Operations – 1

We can use combined operations to determine a more complex transformation.

Reflect about a line thru the origin:

- know how to reflect across axes
- transform line onto x-axis then reflect



13

Combined Operations – 2

1. rotate $(-\Theta)$

$$\mathbf{T}_1 = \frac{1}{\sqrt{m^2+1}}\begin{bmatrix} 1 & m \\ -m & 1 \end{bmatrix}$$

$$\sin\Theta = \frac{m}{\sqrt{m^2+1}} \qquad \cos\Theta = \frac{1}{\sqrt{m^2+1}}$$

2. reflect about $x-$axis

$$\mathbf{T}_2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

3. rotate back (Θ)

$$\mathbf{T}_3 = \frac{1}{\sqrt{m^2+1}}\begin{bmatrix} 1 & -m \\ m & 1 \end{bmatrix}$$

14

Combined Operations – 3

Total transformation : $T = T_3\, T_2\, T_1$

$$T = \frac{1}{m^2+1}\begin{bmatrix} 1 & -m \\ m & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 1 & m \\ -m & 1 \end{bmatrix}$$

$$= \frac{1}{m^2+1}\begin{bmatrix} -m^2+1 & 2m \\ 2m & m^2-1 \end{bmatrix}$$

let the line be $x = y \Rightarrow m = 1$:

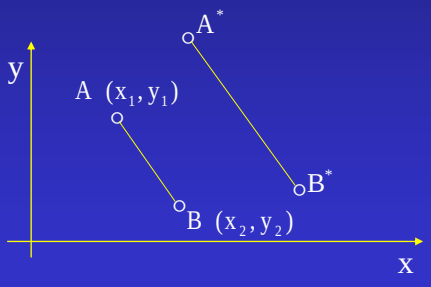
$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

15

Transformation of Lines – 1

Two point define a line,
to transform a line segment → transform end points

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} = \begin{bmatrix} ax_1+by_1 & ax_2+by_2 \\ cx_1+dy_1 & cx_2+dy_2 \end{bmatrix}$$



Notice every point on *AB* has a corresponding transformed point on *A*B**

16

Transformation of Lines – 2

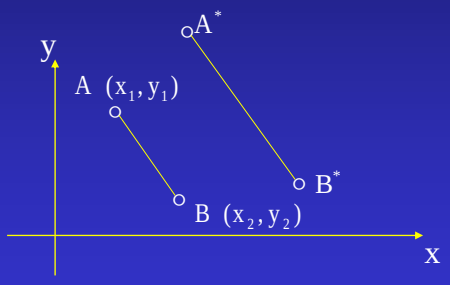
Properties of linear transformations of lines:

1. Straight lines remain straight.
2. Parallel lines remain parallel.
3. Midpoint of line transforms to midpoint.
4. Point of intersection of two lines transforms to point of intersection of transformed lines.

17

Parallel Lines – 1

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} = \begin{bmatrix} ax_1 + by_1 & ax_2 + by_2 \\ cx_1 + dy_1 & cx_2 + dy_2 \end{bmatrix}$$



$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$m^* = \frac{(cx_2 + dy_2) - (cx_1 + dy_1)}{(ax_2 + by_2) - (ax_1 + by_1)}$$

18

Parallel Lines – 2

$$m^* = \frac{(cx_2 + dy_2) - (cx_1 + dy_1)}{(ax_2 + by_2) - (ax_1 + by_1)}$$

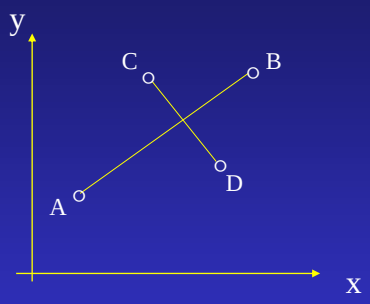
$$= \frac{c(x_2 - x_1) + d(y_2 - y_1)}{a(x_2 - x_1) + b(y_2 - y_1)}$$

$$= \frac{c + d\dfrac{(y_2 - y_1)}{(x_2 - x_1)}}{a + b\dfrac{(y_2 - y_1)}{(x_2 - x_1)}} = \frac{c + dm}{a + bm}$$

Notice the transformed slope, m*, depends only on original slope – not endpoints. So parallel lines transform to parallel lines.

19

Perpendicular Lines



$$m_{CD} = \frac{-1}{m_{AB}} \qquad \therefore m_{AB} m_{CD} = -1$$

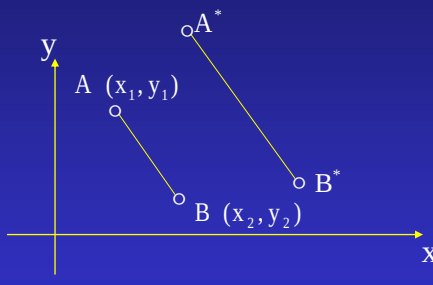
$$m^*_{AB} = \frac{c + dm_{AB}}{a + bm_{AB}}$$

$$m^*_{CD} = \frac{c + dm_{CD}}{a + bm_{CD}} = \frac{c + d\dfrac{(-1)}{m_{AB}}}{a + b\dfrac{(-1)}{m_{AB}}}$$

Ingeneral, $m^*_{AB} m^*_{CD} \neq -1$. Therefore perpendicu lar lines do not remain perpendicu lar after transforma tion.

20

Midpoint – 1



$$\text{midpoint of } AB = \begin{bmatrix} \frac{x_1 + x_2}{2} \\ \frac{y_1 + y_2}{2} \end{bmatrix}$$

$$\text{midpoint of } A^*B^* = \begin{bmatrix} \frac{x_1^* + x_2^*}{2} \\ \frac{y_1^* + y_2^*}{2} \end{bmatrix}$$

21

Midpoint – 2

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} \frac{x_1 + x_2}{2} \\ \frac{y_1 + y_2}{2} \end{bmatrix} = \begin{bmatrix} \frac{a}{2}(x_1 + x_2) + \frac{b}{2}(y_1 + y_2) \\ \frac{c}{2}(x_1 + x_2) + \frac{d}{2}(y_1 + y_2) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2}(ax_1 + by_1 + ax_2 + by_2) \\ \frac{1}{2}(cx_1 + dy_1 + cx_2 + dy_2) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{x_1^* + x_2^*}{2} \\ \frac{y_1^* + y_2^*}{2} \end{bmatrix}$$

22

Pure Translation

So far, all transformations have rotated, stretched, reflected, etc, but none have accomplished pure translation.

All transformations have yielded:

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

Can not get something like:

$$\begin{bmatrix} ax+n \\ dy+m \end{bmatrix}$$

Also, recall that the origin is invariant under this type of transformation.

23

Homogeneous Coordinates – 1

Homogeneous coordinate representation allows transformation of n–dimensional vectors in (n+1)–dimensional space. The transformed n–dimensional results are obtained by projecting the transformed (n+1)–dimensional point back into the n–dimensional space of interest.

ordinary coordinate

$$\begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow$$

homogeneous coordinate

$$\begin{bmatrix} hx \\ hy \\ h \end{bmatrix} = \begin{bmatrix} X \\ Y \\ H \end{bmatrix}$$

24

Homogeneous Coordinates – 2

Homogeneou s coordinates are not unique.

$$\begin{bmatrix}12\\8\\4\end{bmatrix}, \begin{bmatrix}6\\4\\2\end{bmatrix} \text{ and } \begin{bmatrix}3\\2\\1\end{bmatrix}$$

all represent the same cartesian coordinate :

$$\begin{bmatrix}3\\2\end{bmatrix}$$

25

Homogeneous Coordinates – 3

The cartesian coordinate x-y plane can be represented by the h=1 plane in a homogeneous coordinate representation. To obtain the cartesian coordinate which is represented by a homogeneous coordinate divide each entry by h.

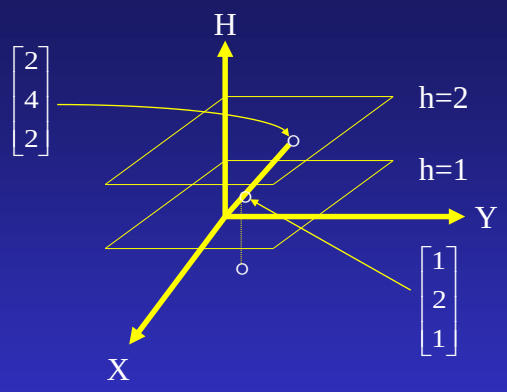
homogeneous cartesian

$$\begin{bmatrix}2\\4\\2\end{bmatrix} = \begin{bmatrix}1\\2\\1\end{bmatrix} = \begin{bmatrix}1\\2\end{bmatrix}$$

This can be viewed as a projection onto the h=1 plane.

26

Homogeneous Coordinates – 4



- Thus there are infinite number of homogeneous coordinates which represent a given point in cartesian coordinates.

27

Homogeneous Coordinates – 5

Consider the homogeneous coordinate:

$\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$ by definition the ordinary cartesian coordinate is:

$\begin{bmatrix} \frac{x}{0} \\ \frac{y}{0} \end{bmatrix} \Rightarrow \begin{bmatrix} \infty \\ \infty \end{bmatrix}$ the point at infinity!

This can be thought of as the point on the vector $\begin{bmatrix} x \\ y \end{bmatrix}$ at ∞

Recall – computers don't like infinity so that gives a method to handle it.

28

General 2 – D transformations

$$\begin{bmatrix} a & b & m \\ c & d & n \\ p & q & s \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax+by+m \\ cx+dy+n \\ px+qy+s \end{bmatrix}$$

Thus the ordinary phsical coordinates :

$$x^* = \frac{ax+by+m}{px+qy+s} \qquad y^* = \frac{cx+dy+n}{px+qy+s}$$

Four of these element a, b, c and d have all ready been discussed. The remaining elements will be considered next.

29

Pure Translations

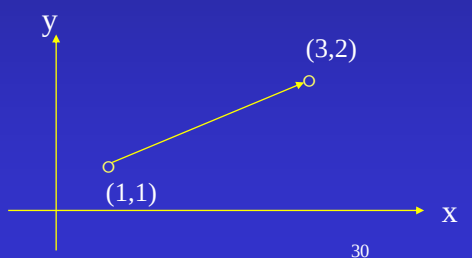
This 3 dimensional transformation allows us to perform translation in 2 – space.

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{m} \\ \mathbf{0} & \mathbf{1} & \mathbf{n} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{x+m} \\ \mathbf{y+n} \\ \mathbf{1} \end{bmatrix}$$

Example: want to move the point (1,1) to (3,2)

i.e. 2 units in x and y unit in y

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$



30

Overall Scaling

The s term controls overall uniform scaling :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ s \end{bmatrix} \quad \Rightarrow \quad x^* = \frac{x}{s}, \quad y^* = \frac{y}{s}$$

if $s < 1$ ⇨ enlarge
$s > 1$ ⇨ reduce

distinct from scaling with a & d, these are called "local" scaling

All points end up on the **h**=s plane must projected back onto **h**=1 plane

Projection

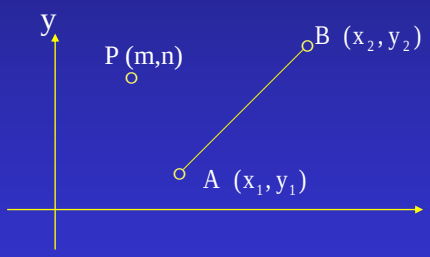
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p & q & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ px+qy+1 \end{bmatrix} \quad \Rightarrow \quad x^* = \frac{x}{px+qy+1}, \quad y^* = \frac{y}{px+qy+1}$$

So, a transformed point lands on the plane **h**=p**x**+q**y**+1 and must be projected back onto **h**=1.

i.e. each (**x**,**y**) ends up on a different h plane

Combined Operations – 1

Consider rotation about a point other than the origin.



Steps :

1. translate **P** to origin
2. rotate **AB** about **P**
3. translate back

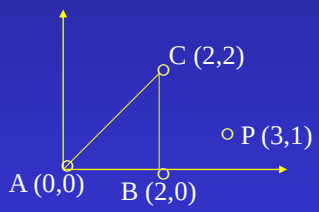
33

Combined Operations – 2

$$T = \begin{bmatrix} 1 & 0 & m \\ 0 & 1 & n \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -m \\ 0 & 1 & -n \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\Theta & -\sin\Theta & -m(\cos\Theta - 1) + n\sin\Theta \\ \sin\Theta & \cos\Theta & -m\sin\Theta - n(\cos\Theta - 1) \\ 0 & 0 & 1 \end{bmatrix}$$

Example:



rotate 90°

about **P**

34

Combined Operations – 3

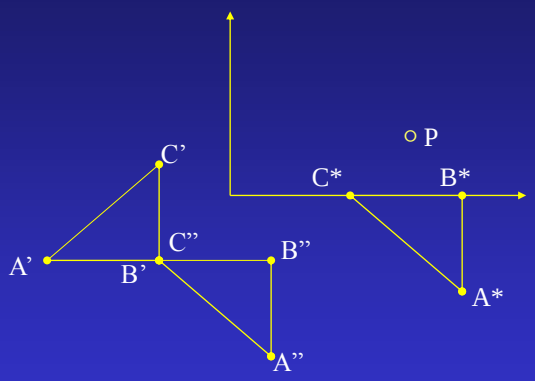
$$\text{1.}\begin{bmatrix}1 & 0 & -3\\ 0 & 1 & -1\\ 0 & 0 & 1\end{bmatrix}\begin{matrix}A\\ \\ \\ \end{matrix}\begin{matrix}B & C\\ \\ \\ \end{matrix}\begin{bmatrix}0 & 2 & 2\\ 0 & 1 & 2\\ 1 & 1 & 1\end{bmatrix}=\begin{matrix}A' & B' & C'\\ \\ \\ \end{matrix}\begin{bmatrix}-3 & -1 & -1\\ -1 & -1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$$\text{2.}\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}-3 & -1 & -1\\ -1 & -1 & 1\\ 1 & 1 & 1\end{bmatrix}=\begin{matrix}A'' & B'' & C''\\ \\ \\ \end{matrix}\begin{bmatrix}1 & 1 & -1\\ -3 & -1 & -1\\ 1 & 1 & 1\end{bmatrix}$$

$$\text{3.}\begin{bmatrix}1 & 0 & 3\\ 0 & 1 & 1\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 1 & -1\\ -3 & -1 & -1\\ 1 & 1 & 1\end{bmatrix}=\begin{matrix}A^* & B^* & C^*\\ \\ \\ \end{matrix}\begin{bmatrix}4 & 4 & 2\\ -2 & 0 & 0\\ 1 & 1 & 1\end{bmatrix}$$

35

Combined Operations – 4



36

Conclusion

$$T = \left[\begin{array}{cc|c} a & b & m \\ c & d & n \\ \hline p & q & s \end{array}\right]$$

$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \Rightarrow$ local scaling, shear, rotation about origin (stretching)

$\begin{bmatrix} m \\ n \end{bmatrix} \Rightarrow$ translation

$\begin{bmatrix} s \end{bmatrix} \Rightarrow$ overall scaling

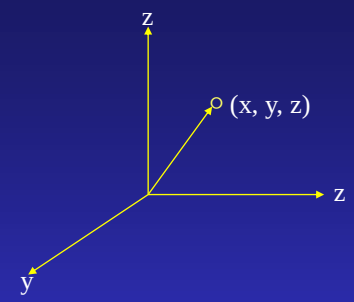
$\begin{bmatrix} p & q \end{bmatrix} \Rightarrow$ projection

37

3 - D Transformation

38

4 – D Homogeneous Coordinates



cartesian

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

homogeneous

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

$$\begin{bmatrix} a & b & c & l \\ d & e & f & m \\ g & h & i & n \\ p & q & r & s \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix}$$

project back onto $h = 1$

$\Rightarrow$

$$\begin{bmatrix} x^* = \frac{X}{H} \\ y^* = \frac{Y}{H} \\ z^* = \frac{Z}{H} \\ 1 \end{bmatrix}$$

39

General 3 – D Transformation

$$T = \left[\begin{array}{ccc:c} a & b & c & l \\ d & e & f & m \\ g & h & i & n \\ \hdashline p & q & r & s \end{array}\right]$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \Rightarrow$$ **local scaling, shear, rotate and reflect**

$$\begin{bmatrix} l \\ m \\ n \end{bmatrix} \Rightarrow$$ **translate**

$$\begin{bmatrix} p & q & r \end{bmatrix} \Rightarrow$$ **perspective transformation**

$$\begin{bmatrix} s \end{bmatrix} \Rightarrow$$ **overall scaling**

40

Local Scaling

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ ey \\ iz \\ 1 \end{bmatrix}$$

41

Overall Scaling

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} \frac{x}{s} \\ \frac{y}{s} \\ \frac{z}{s} \\ 1 \end{bmatrix}$$

$s < 1 \Rightarrow$ enlarge

$s > 1 \Rightarrow$ reduce

42

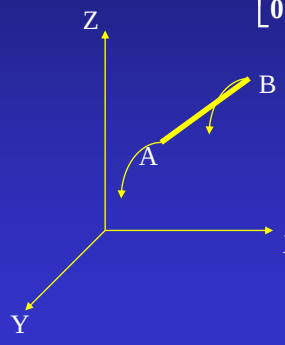
Shear

$$\begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ g & h & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + by + cz \\ y + dx + fz \\ z + gx + hy \\ 1 \end{bmatrix}$$

43

Rotation about the x–axis

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



x coordinate stays the same

44

Rotation about the y – axis

$$T = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

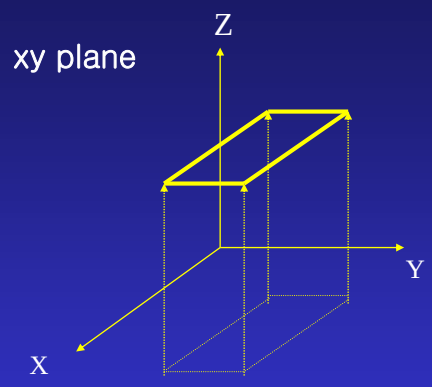
45

Rotation about the z – axis

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

46

Reflection thru Coordinate Plane



$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note: other reflections are similar

47

Translation

$$\begin{bmatrix} 1 & 0 & 0 & l \\ 0 & 1 & 0 & m \\ 0 & 0 & 1 & n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+l \\ y+m \\ z+n \\ 1 \end{bmatrix}$$

48

Combined Operation – 1

Reflect thru z = 2 plane.

1. translate z = 2 to z = 0 (xy plane)
2. reflect thru xy plane
3. translate back

$$T = \underset{T_3}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}} \underset{T_2}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}} \underset{T_1}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

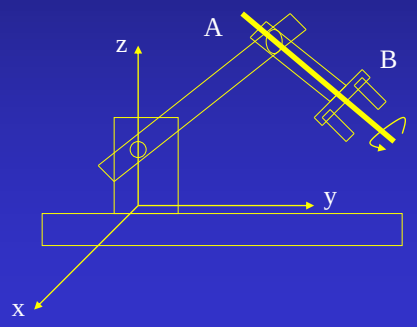
Combined Operation – 2

Rotate about a line parallel to the z – axis thru the point (l, m, n)

$$T = \underset{T_3}{\begin{bmatrix} 1 & 0 & 0 & l \\ 0 & 1 & 0 & m \\ 0 & 0 & 1 & n \\ 0 & 0 & 0 & 1 \end{bmatrix}} \underset{T_2}{\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}} \underset{T_1}{\begin{bmatrix} 1 & 0 & 0 & -l \\ 0 & 1 & 0 & -m \\ 0 & 0 & 1 & -n \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

Rotation about Arbitrary Axes – 1

Rotation about an arbitrary axes in space are frequently found in engineer application.



51

Rotation about Arbitrary Axes – 2

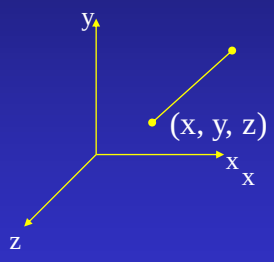
Sequence of steps:

1. Translate the arbitrary axis so that one of its end and points coincides with the origin.
2. Rotate about the x and y axes to align the arbitrary axis with the positive z – axis.
3. Rotate about the z – axis by the desired angle, θ
4. Apply reverse rotations about y and x axes to bring the arbitrary axis back to its original position with respect to the origin.
5. Apply reverse translations to place the arbitrary axis back in its initial position.

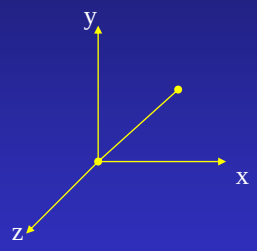
52

Step 1

Translate the arbitrary axis so that one of its end points coincides with the origin.



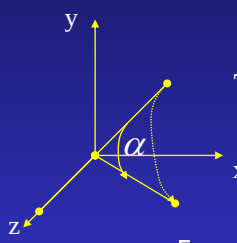
$$\mathbf{T}_{tr} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



53

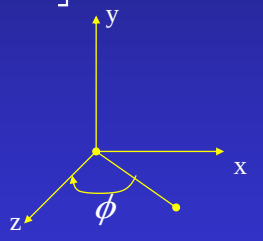
Step 2

Rotate about the x and y axes to align the arbitrary axis with the positive z – axis.



$$\mathbf{T}_{rx}(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}_{ry}(\varphi) = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



54

Step 3

Rotate about the z – axis by the desired angle, θ

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

55

Concatenation

Steps 4 and 5 use the similar transformations as steps 2 and 1 respectively.

$$T_{rarb} =$$
$$T_{tr}(x, y, z)\, T_{rx}(-\alpha)\, T_{ry}(-\varphi)\, T_{rz}(\theta)\, T_{ry}(\varphi)\, T_{rx}(\alpha)\, T_{tr}(-x, -y, -z)$$

56

3D Shearing in Computer Graphics | Definition | Examples

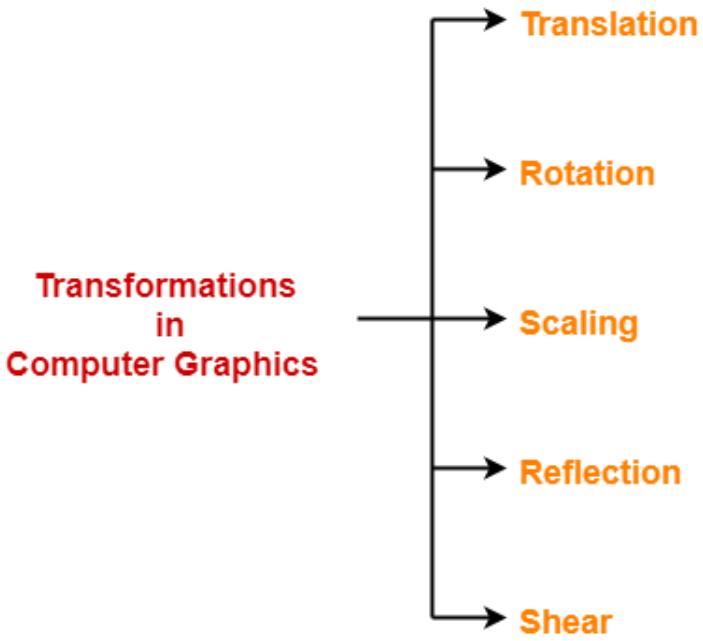
Computer Graphics

3D Transformations in Computer Graphics-

We have discussed-

- Transformation is a process of modifying and re-positioning the existing graphics.
- 3D Transformations take place in a three dimensional plane.

In computer graphics, various transformation techniques are-



1. **Translation**
2. **Rotation**
3. **Scaling**
4. **Reflection**
5. **Shear**

In this article, we will discuss about 3D Shearing in Computer Graphics.

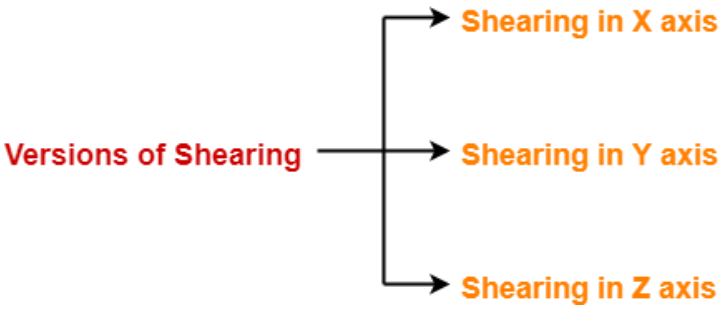
3D Shearing in Computer Graphics-

In Computer graphics,

3D Shearing is an ideal technique to change the shape of an existing object in a three dimensional plane.

In a three dimensional plane, the object size can be changed along X direction, Y direction as well as Z direction.

So, there are three versions of shearing-



1. Shearing in X direction
2. Shearing in Y direction
3. Shearing in Z direction

Consider a point object O has to be sheared in a 3D plane.

Let-

- Initial coordinates of the object O = (X_{old}, Y_{old}, Z_{old})
- Shearing parameter towards X direction = Sh_x

- Shearing parameter towards Y direction = Sh_y
- Shearing parameter towards Z direction = Sh_z
- New coordinates of the object O after shearing = $(X_{new}, Y_{new}, Z_{new})$

Shearing in X Axis-

Shearing in X axis is achieved by using the following shearing equations-

- $X_{new} = X_{old}$
- $Y_{new} = Y_{old} + Sh_y \times X_{old}$
- $Z_{new} = Z_{old} + Sh_z \times X_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ Sh_y & 1 & 0 & 0 \\ Sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Shearing Matrix
(In X axis)

Shearing in Y Axis-

Shearing in Y axis is achieved by using the following shearing equations-

- $X_{new} = X_{old} + Sh_x \times Y_{old}$
- $Y_{new} = Y_{old}$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & Sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Shearing Matrix

(In Y axis)

Shearing in Z Axis-

Shearing in Z axis is achieved by using the following shearing equations-

- $X_{new} = X_{old} + Sh_x \times Z_{old}$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old}$
- $Z_{new} = Z_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & Sh_x & 0 \\ 0 & 1 & Sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Shearing Matrix

(In Z axis)

PRACTICE PROBLEMS BASED ON 3D SHEARING IN COMPUTER GRAPHICS-

Problem-01:

Given a 3D triangle with points (0, 0, 0), (1, 1, 2) and (1, 1, 3). Apply shear parameter 2 on X axis, 2 on Y axis and 3 on Z axis and find out the new coordinates of the object.

Solution-

Given-

- Old corner coordinates of the triangle = A (0, 0, 0), B(1, 1, 2), C(1, 1, 3)
- Shearing parameter towards X direction (Sh_x) = 2
- Shearing parameter towards Y direction (Sh_y) = 2
- Shearing parameter towards Y direction (Sh_z) = 3

Shearing in X Axis-

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 0$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 0 = 0$
- $Z_{new} = Z_{old} + Sh_z \times X_{old} = 0 + 3 \times 0 = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$
- $Z_{new} = Z_{old} + Sh_z \times X_{old} = 2 + 3 \times 1 = 5$

Thus, New coordinates of corner B after shearing = (1, 3, 5).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = (X_{new}, Y_{new}, Z_{new}).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 1$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$
- $Z_{new} = Z_{old} + Sh_z \times X_{old} = 3 + 3 \times 1 = 6$

Thus, New coordinates of corner C after shearing = (1, 3, 6).

Thus, New coordinates of the triangle after shearing in X axis = A (0, 0, 0), B(1, 3, 5), C(1, 3, 6).

Shearing in Y Axis-

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = $(X_{new}, Y_{new}, Z_{new})$.

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 0 + 2 \times 0 = 0$
- $Y_{new} = Y_{old} = 0$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old} = 0 + 3 \times 0 = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = $(X_{new}, Y_{new}, Z_{new})$.

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 1 + 2 \times 1 = 3$
- $Y_{new} = Y_{old} = 1$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old} = 2 + 3 \times 1 = 5$

Thus, New coordinates of corner B after shearing = (3, 1, 5).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = $(X_{new}, Y_{new}, Z_{new})$.

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Y_{old} = 1 + 2 \times 1 = 3$
- $Y_{new} = Y_{old} = 1$
- $Z_{new} = Z_{old} + Sh_z \times Y_{old} = 3 + 3 \times 1 = 6$

Thus, New coordinates of corner C after shearing = (3, 1, 6).

Thus, New coordinates of the triangle after shearing in Y axis = A (0, 0, 0), B(3, 1, 5), C(3, 1, 6).

Shearing in Z Axis-

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = $(X_{new}, Y_{new}, Z_{new})$.

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Z_{old} = 0 + 2 \times 0 = 0$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old} = 0 + 2 \times 0 = 0$
- $Z_{new} = Z_{old} = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = $(X_{new}, Y_{new}, Z_{new})$.

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Z_{old} = 1 + 2 \times 2 = 5$

- $Y_{new} = Y_{old} + Sh_y \times Z_{old} = 1 + 2 \times 2 = 5$
- $Z_{new} = Z_{old} = 2$

Thus, New coordinates of corner B after shearing = (5, 5, 2).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = $(X_{new}, Y_{new}, Z_{new})$.

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Z_{old} = 1 + 2 \times 3 = 7$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old} = 1 + 2 \times 3 = 7$
- $Z_{new} = Z_{old} = 3$

Thus, New coordinates of corner C after shearing = (7, 7, 3).

Thus, New coordinates of the triangle after shearing in Z axis = A (0, 0, 0), B(5, 5, 2), C(7, 7, 3).

To gain better understanding about 3D Shearing in Computer Graphics,

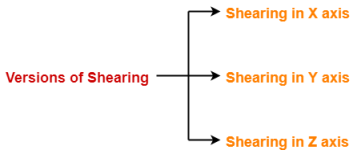
Watch this Video Lecture

Next Article- Bezier Curves

Get more notes and other study material of **Computer Graphics**.

Watch video lectures by visiting our YouTube channel **LearnVidFun.**

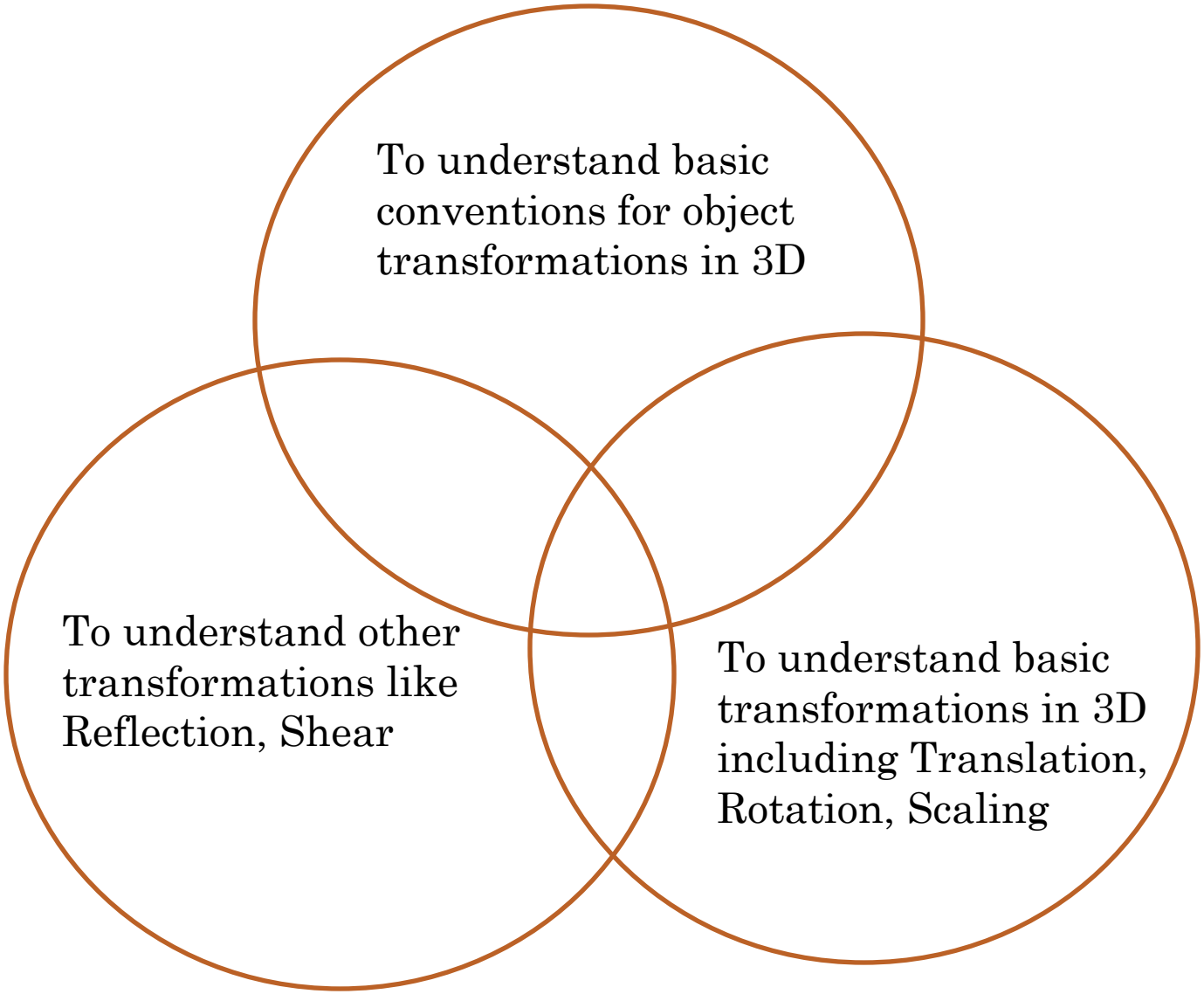
Summary



Article Name	3D Shearing in Computer Graphics | Definition | Examples
Description	3D Shearing in Computer Graphics is a process of modifying the shape of an object in 3D plane. Shearing Transformation in Computer Graphics Definition, Solved Examples and Problems.
Author	Akshay Singhal
Publisher Name	Gate Vidyalay
Publisher Logo	GateVidyalay A temple of learning

CONTENTS

OBJECTIVE


To understand basic conventions for object transformations in 3D
To understand other transformations like Reflection, Shear
To understand basic transformations in 3D including Translation, Rotation, Scaling

TRANSFORMATION

- Transformations are a fundamental part of the computer graphics. Transformations are the movement of the object in Cartesian plane .
- TYPES OF TRANSFORMATION
- There are two types of transformation in computer graphics.

  1) 2D transformation

  2) 3D transformation
- Types of 2D and 3D transformation

  1) Translation 2) Rotation 3) Scaling 4) Shearing 5) reflection

3D Transformation

- When the transformation takes place on a 3D plane, it is called 3D transformation
- The translation, scaling and rotation transformations used for 2D can be extended to three dimensions.
- In 3D, each transformation is represented by a 4x4 matrix.
- Using homogeneous coordinates it is possible to represent each type of transformation in a matrix form and integrate transformations into one matrix
- To apply transformations, simply multiply matrices, also easier in hardware and software implementation
- Homogeneous coordinates can represent directions

- Homogeneous coordinates: 4 components
- Transformation matrices: 4×4 elements

$$\begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- WHY WE USE TRANSFORMATION

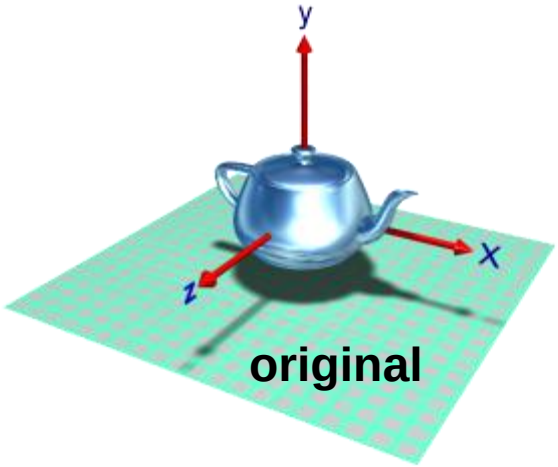
  Transformation are used to position objects, to shape object, to change viewing positions, and even how something is viewed.

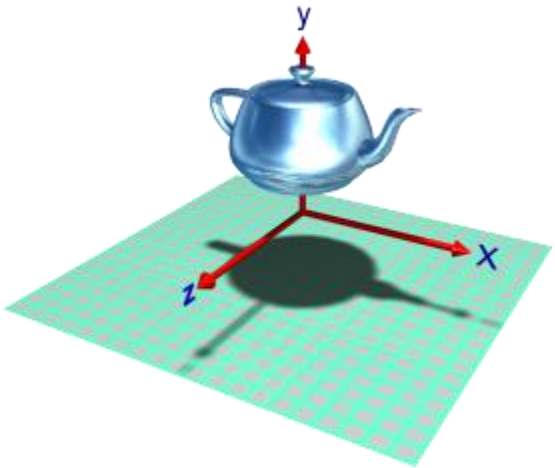
3D TRANSLATION

- Moving of object is called translation.
- In 3 dimensional homogeneous coordinate representation , a point is transformed from position P = (x, y , z) to P'=(x', y', z')

  This can be written as: Using P'= T . P where

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



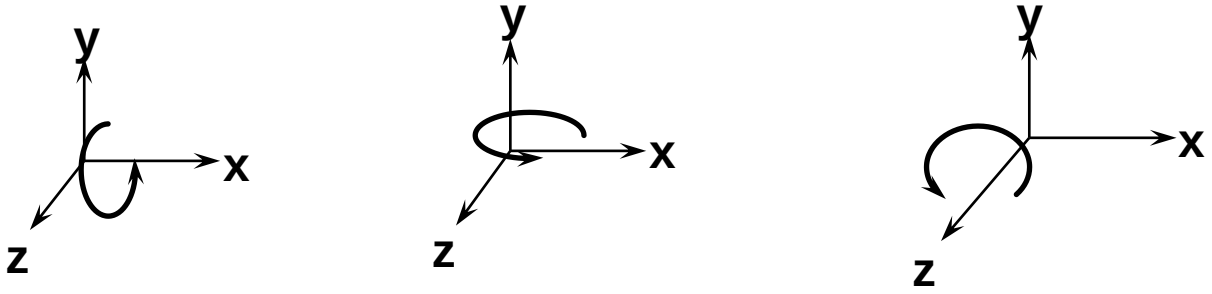


translation along y,
or V = (0, k, 0)

TRANSLATION

3-D Rotations

- Rotation needs an angle and an axis.
- Rotation is defined according to the right-hand rule (our convention)
- In 3D, rotation is about a vector, which can be done through rotations about x, y or z axes.
- Positive rotations are anti-clockwise, negative rotations are clockwise, when looking down a positive axis towards the origin



3-D ROTATION TRANSFORMATION MATRIX

Rotations are orthogonal matrices, preserving distances and angles.

1. Rotation about X-axis

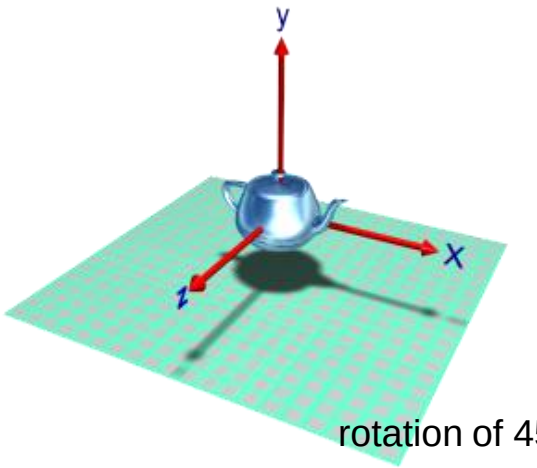
$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

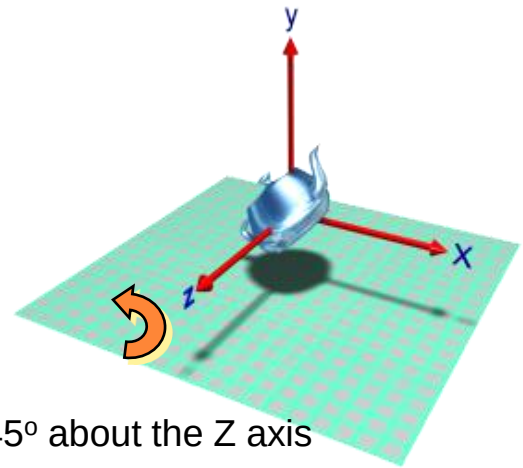
2. Rotation about Y-axis

$$\mathbf{R}_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

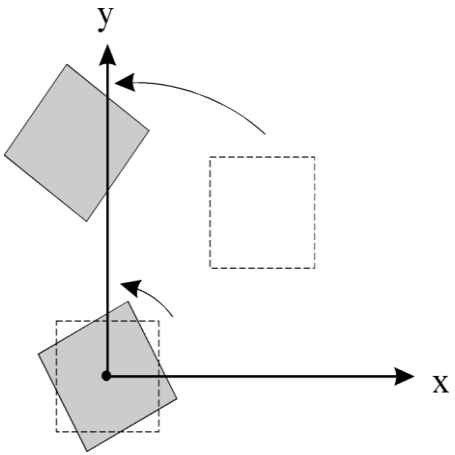
3. Rotation about Z-axis

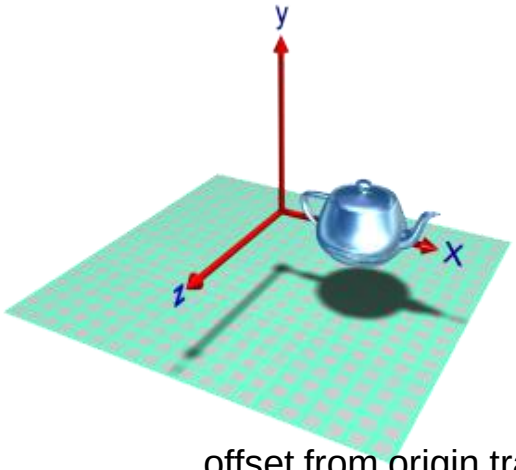
$$\mathbf{R}_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

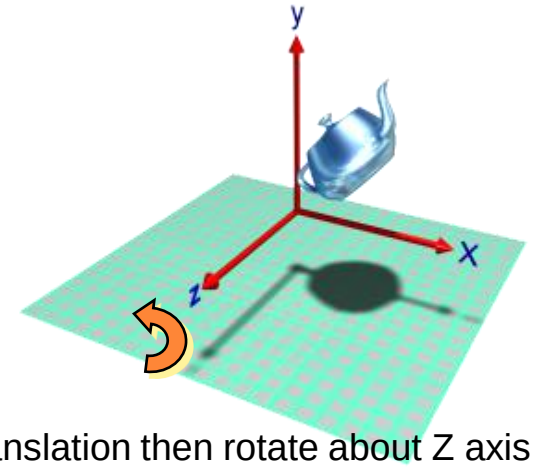




rotation of 45^{o} about the Z axis







offset from origin translation then rotate about Z axis

Rotation

3-D SCALING

- You can change the size of an object using scaling transformation .
- In the scaling process , you either expand or compress the dimensions of the object .
- Scaling can be achieved by multiplying the original coordinates of the object with scaling factor to get the desired result
- The general transformation matrix for scaling is

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} t_x & 0 & 0 & 0 \\ 0 & t_y & 0 & 0 \\ 0 & 0 & t_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3-D SCALING

- Scaling in x-direction by factor a

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scaling in y-direction by factor e

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scaling in z-direction by factor j

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3-D SCALING

- Scaling in x,y,z-direction by factor a,e, j-units respectively.

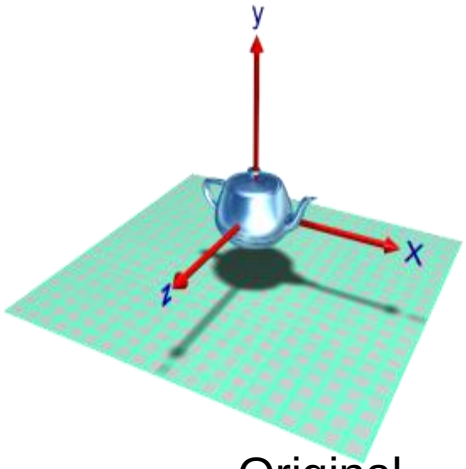
$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Uniform scaling in x-direction by factor a, if a>1 then expansion occurs if 0<a<1 contraction occurs

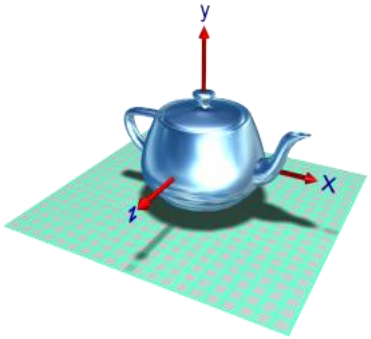
$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Overall scaling by s units
  Here, a=1/s
  if 0<s<1 then expansion occurs
  if s>1 then contraction occurs

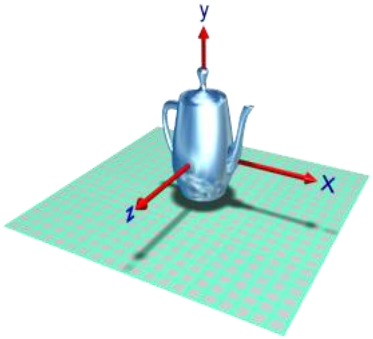
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix}$$



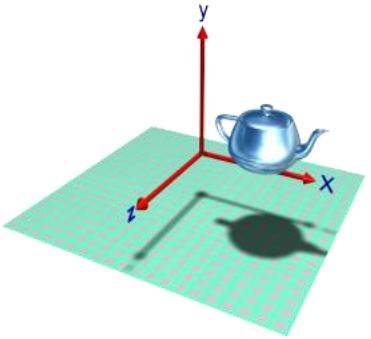
Original



scale all axes



scale Y axis



offset from origin

Scaling

3D REFLECTION

- Reflection in computer graphics is used to emulate reflective objects like mirrors and shiny surfaces
- Reflection may be an x-axis y-axis , z-axis. and also in the planes xy-plane,yz-plane , and zx-plane.
- **Note:** Reflection relative to a given Axis are equivalent to 180 Degree rotations

3D REFLECTION

- Reflection about xy-plane(or z=0plane)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Reflection about yz-plane(or x=0plane)

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D REFLECTION

- Reflection about zx-plane(or y=0plane)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Shearing

- Shearing:

  The change in each coordinate is a linear combination of all three

- The general Transformation matrix for shearing in x, y and z-direction is given by,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Shearing

- Shearing in x-direction proportional to y and z-axis by c-units and e-units respectively is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ c & 1 & 0 & 0 \\ e & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

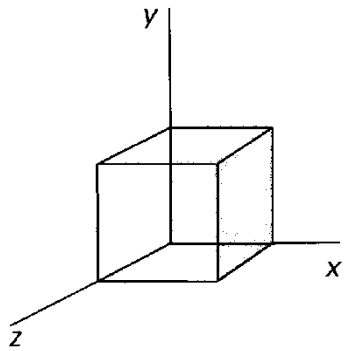
- Shearing in y-direction proportional to x and z-axis by a-units and f-units respectively is

$$\begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

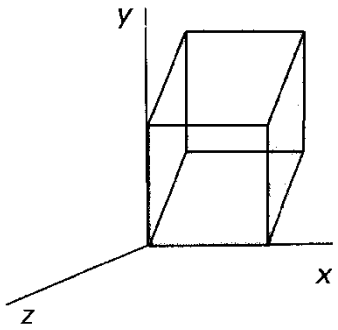
3D Shearing

- Shearing in z-direction proportional to x and y-axis by b-units and d-units respectively is

$$\begin{bmatrix} 1 & 0 & b & 0 \\ 0 & 1 & d & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Original object



Shearing in z-direction

Summary

- Methods for object modelling transformation in three dimensions are extended from two dimensional methods by including consideration for the z coordinate.
- We saw transformation in 3-D using which we can transformed the required object in required form.

Virtual and Augmented Reality
Introduction to Computer Graphics Programming

Pavan Kumar B N,
CSE Group, IIITS

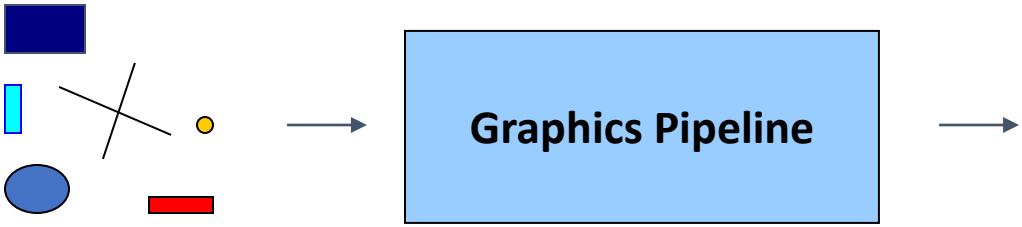
Graphics Programming

OpenGL

- Low-level API
- cross-language
- cross-platform
- 2D, 3D computer graphics

OpenGL Basics

- Rendering
  - Typically execution of OpenGL commands
  - Converting geometric/mathematical object descriptions into frame buffer values
- OpenGL can render:
  - Geometric primitives
    - Lines, points, polygons, etc…
  - Bitmaps and Images
    - Images and geometry linked through texture mapping



OpenGL and GLUT

- GLUT (OpenGL Utility Toolkit)
  - An auxiliary library
    - A portable windowing API
    - Easier to show the output of your OpenGL application
    - Not officially part of OpenGL
  - Handles:
    - Window creation,
    - OS system calls
      - Mouse buttons, movement, keyboard, etc…
    - Callbacks

How to install GLUT?

- Download GLUT
  - http://www.opengl.org/resources/libraries/glut.html
- Copy the files to following folders:
  - glut.h → VC/include/gl/
  - glut32.lib → VC/lib/
  - glut32.dll → windows/system32/
- Header Files:
  - #include <GL/glut.h>
  - #include <GL/gl.h>
  - Include glut automatically includes other header files

GLUT Basics

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Register input callback functions
    - render
    - resize
    - input: keyboard, mouse, etc.
  - Enter event processing loop

OpenGL - Intro

- Consider program for displaying a straight line - written using OpenGL functions in C
- Let's try to understand

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

OpenGL - Intro

- First thing is to include **header file** containing graphics library functions
- Thus, very first line is

  #include<GL/glut.h>

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (& argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

OpenGL - Intro

- OpenGL core library does not provide support for I/O (library functions designed to be device-independent)
  - But we have to display line (output)

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (& argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

OpenGL - Intro

- To display, auxiliary libraries are required - on top of core library
  - Provided in GLUT or OpenGL Utility Toolkit library

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

OpenGL - Intro

- GLUT provides a library of functions for interacting with *any* screen-windowing system
  - Allow us to set up a *display window* on our screen (a rectangular area showing the line)

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (& argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

- GLUT allows us create and manage **display window** - screen region to display line
- The first thing required is to initialize GLUT - with the statement (in main())

  glutInit (& argc, argv);

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (& argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

- After initialization, we can set various options for display window
  - Using *glutInitDisplayMode* function

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

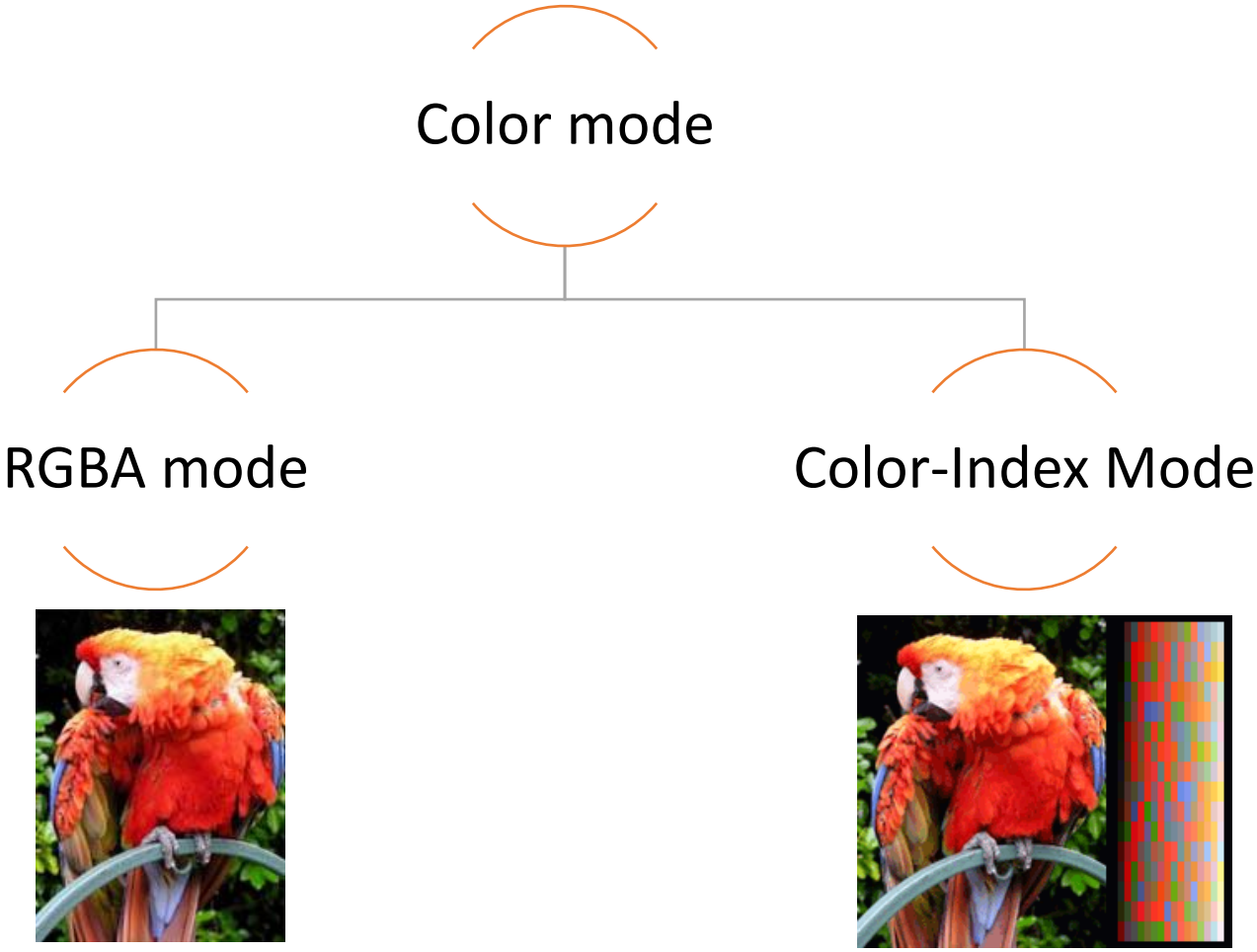
Main() Functions

- Following line of code specifies
  - A single refresh buffer to be used for display window
  - RGB color mode to be used for selecting color values

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Colors: RGBA vs. Color-Index



Main() Functions

- GLUT provides for some default position and size of the display window – can change with

  glutInitWindowPosition (0, 0);

  glutInitWindowSize (800, 600);

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

glutInitWindowPosition (0, 0);

- Specify top-left corner position of window
  - Specified in integer screen coordinates (*X* and *Y*, in that order), assuming origin at top-left corner

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

- Next, we create window and set a caption (optional)

  glutCreateWindow ("The OpenGL example");

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

- Next, we specify picture to be displayed in the window - the line
  - We create this picture in a separate function *createLine()*

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

- *createLine* function passed as argument to *glutDisplayFunc* - indicating line to be displayed on window
  - However, certain initializations are required before we do that

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Init() Function

- Initializations and one-time parameter settings done
- THREE OpenGL library routines called in it

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Init() Function

glClearColor (1.0, 1.0, 1.0, 0.0);

- Used to set a background color to our display window
- Color specified with RGB components

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (& argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Init() Function

R G B

glClearColor (1.0, 1.0, 1.0, 0.0);

- RGB values supplied through first three arguments, in that order
- Here we are setting window background to WHITE
- If we set all components to 0.0, we will get the color black.

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Init() Function

glClearColor (1.0, 1.0, 1.0, 0.0);

- Fourth parameter - *alpha* value for specified color
  - Used as a **blending** parameter - specifying way to color two **overlapping** objects
  - 0.0 → totally transparent and 1.0 → totally opaque objects

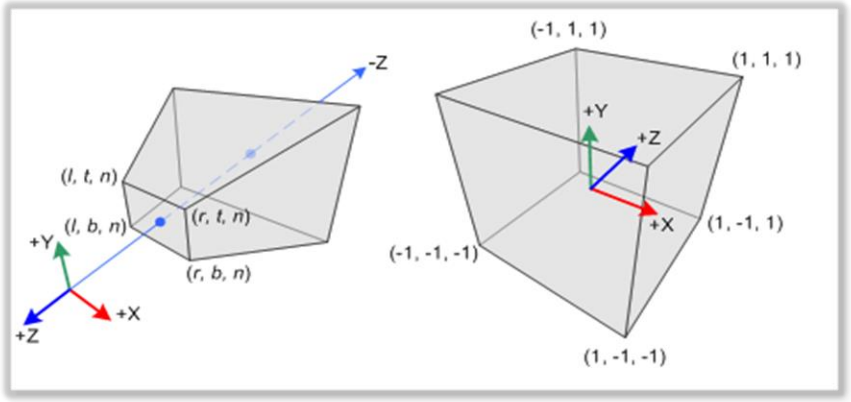
```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

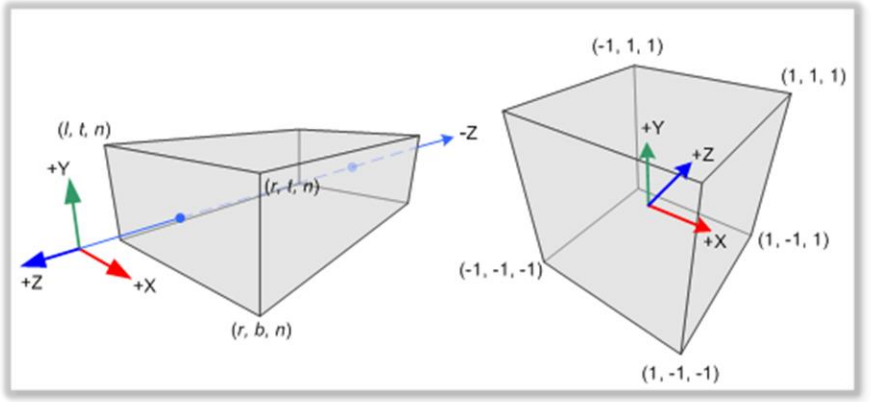
Init() Function

- Thus, need to specify projection type and other viewing parameters
- Done with
  - glMatrixMode (GL_PROJECTION)
  - gluOrhto2D (0.0, 800.0, 0.0, 600.0)

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Projection
Perspective vs. Orthographic





Objects which are far away are smaller than those nearby;

Does not preserve the shape of the objects.

Perspective view points give more information about depth;
Easier to view because you use perspective views in real life.

Useful in architecture, game design, art etc.

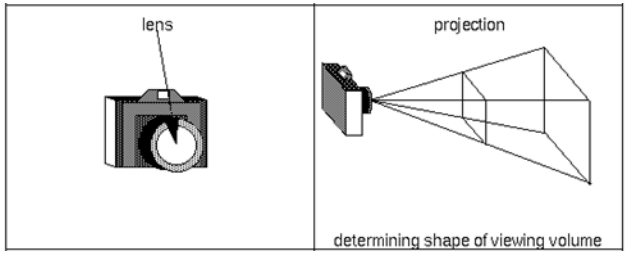
All objects appear the same size regardless the distance;

Orthographic views make it much easier to compare sizes of the objects. It is possible to accurately measure the distances

All views are at the same scale

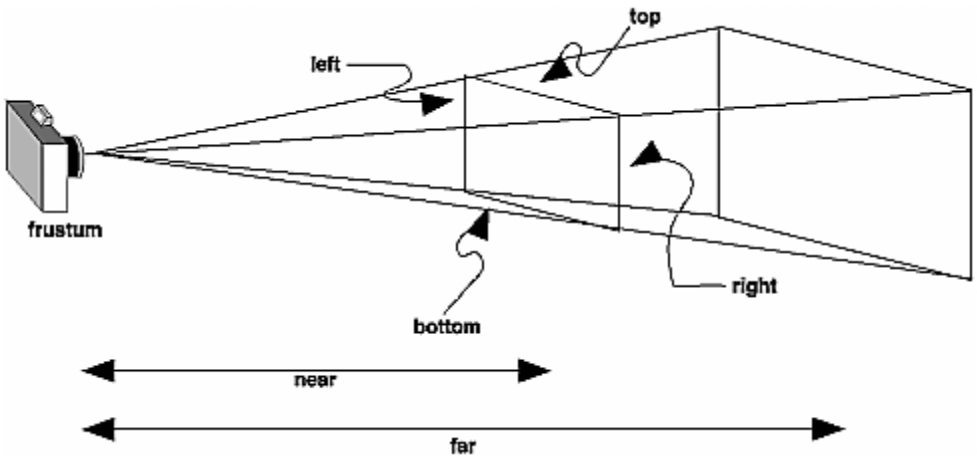
Very useful for cartography, engineering drawings, machine parts.

Projection transformation



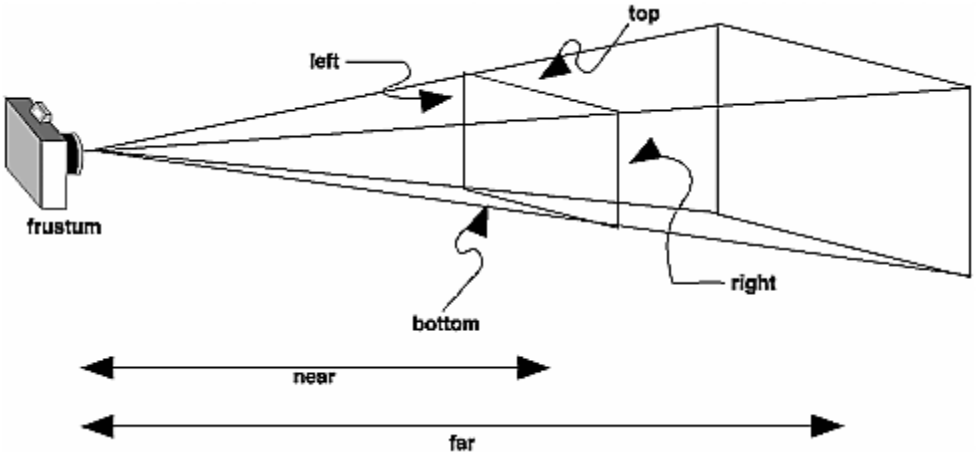

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

//perspective projection
glFrustum(left, right, bottom, top, near, far);

Or

//orthographic projection
glOrtho (left, right, bottom, top, near, far);
```

Perspective Transformation




```
//perspective projection
void glFrustum(double left,
               double right,
               double bottom,
               double top,
               double near,
               double far);
```

Perspective Transformation

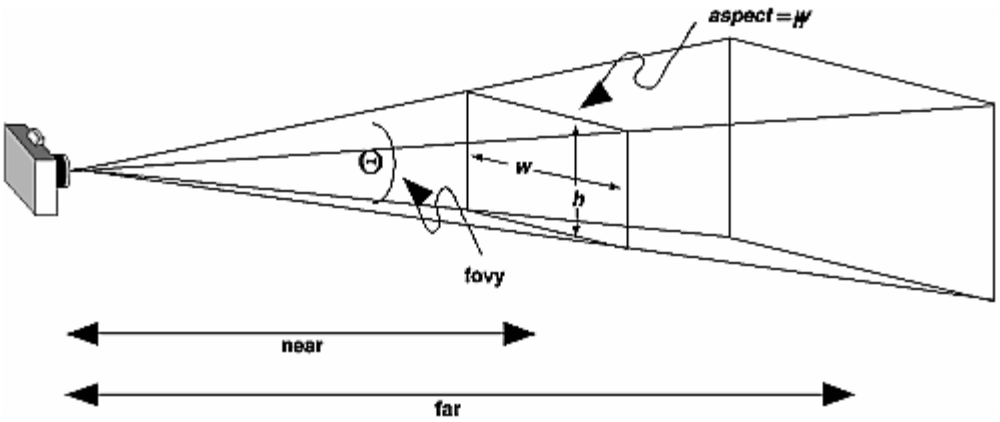


Four sides of the frustum, its top, and its base correspond to the six clipping planes of the viewing volume.

Objects or parts of objects outside these planes are clipped from the final image

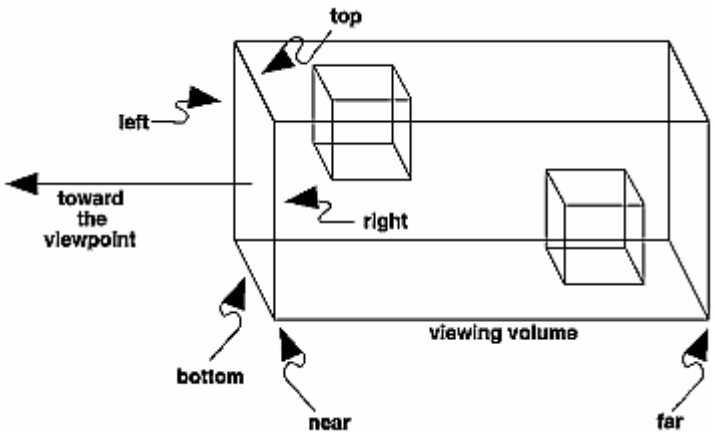
Does not have to be symmetrical

Perspective Transformation




```
//perspective projection
void gluPerspective(  double fovy,
                      double aspect,
                      double near,
                      double far);
```

Orthographic Transformation




```
//orthographic projection
void glOrtho(  double left,
               double right,
               double bottom,
               double top,
               double near,
               double far);
```

Init() Function

gluOrhto2D (0.0, 800.0, 0.0, 600.0)

- 2nd function prefixed with *glu* - indicates it belongs to GLU or OpenGL Utility, an auxiliary library
  - Provides routines for complex tasks - setting up of viewing and projection matrices, describing complex objects with line and polygon approximations, processing surface-rendering operations and displaying splines with linear approximations

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- OpenGL also allows to set object color - with function glColor3f (0.0, 1.0, 0.0);
- 3 arguments specify *RGB* components

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- 3f in function name indicates 3components specified using floating-point values
  - Values can range between 0.0 and 1.0
  - Values in example denote green color

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- Next, piece of code to specify a line segment between end points (200, 100) and (20, 50)

```
glBegin (GL_LINES);
  glVertex2i (200, 100);
  glVertex2i (20, 50);
glEnd ();
```

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- Line end points specified using OpenGL function *glVertex2i*
  - *2i* indicates vertices are specified by two integer values denoting *X* and *Y* coordinates

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- 1st and 2nd end points determined depending on their ordering in the code
  - Here (200, 100) 1st end point while (20, 50) acts as 2nd end point

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100); 1st
    glVertex2i (20, 50);   2nd
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- *glBegin* with its symbolic OpenGL constant GL_LINES along with the function *glEnd* indicate vertices are line end points

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- With all these functions, basic line creation program is ready
  - However, functions may be stored at different locations in memory, depending on OpenGL implementation

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

createLine() Function

- Need to **force** system to process all these functions
  - This we do with glFlush ();
  - Should be the last line of our picture generation procedure

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

- The display window is not yet on the screen - need to *activate* it, once window content decided

  glutMainLoop ();

- Activates **all** display windows created along with their graphic contents.

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Main() Functions

- This function must be the last one in our program
  - It puts program into an infinite loop
  - In this loop, program waits for inputs from devices such as mouse/keyboard
  - Even if no input, loop ensures picture displayed till window closed

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

GLUT primitives

- void glut**SolidSphere**(GLdouble radius, GLint slices, GLint stacks);
- void glut**WireSphere**(GLdouble radius, GLint slices, GLint stacks);
- void glutSolid**Cube**(GLdouble size);
- void glutSolid**Cone**(GLdouble base, GLdouble height, GLint slices, GLint stacks);
- void glutSolid**Torus**(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
- void glutSolid**Dodecahedron**(void); // radius sqrt(3)
- void glutSolid**Tetrahedron**(void); // radius sqrt(3)
- void glutSolid**Icosahedron**(void) // radius 1
- void glutSolid**Octahedron**(void); // radius 1

GLUT primitives

- `glutSolidSphere` `glutWireSphere`
- `glutSolidCube` `glutWireCube`
- `glutSolidCone` `glutWireCone`
- `glutSolidTorus` `glutWireTorus`
- `glutSolidDodecahedron` `glutWireDodecahedron`
- `glutSolidOctahedron` `glutWireOctahedron`
- `glutSolidTetrahedron` `glutWireTetrahedron`
- `glutSolidIcosahedron` `glutWireIcosahedron`
- `glutSolidTeapot` `glutWireTeapot`

- More info: http://www.opengl.org/documentation/specs/glut/spec3/node80.html

Additional GLUT callback routines

GLUT supports many different callback actions, including:

glutDisplayFunc() defines the function that sets up the image on the screen

glutReshapeFunc() function is called when the size of the window is changed

glutKeyBoardFunc() callback routine to respond on keyboard entry

glutMouseFunc() callback to respond on pressing the mouse button

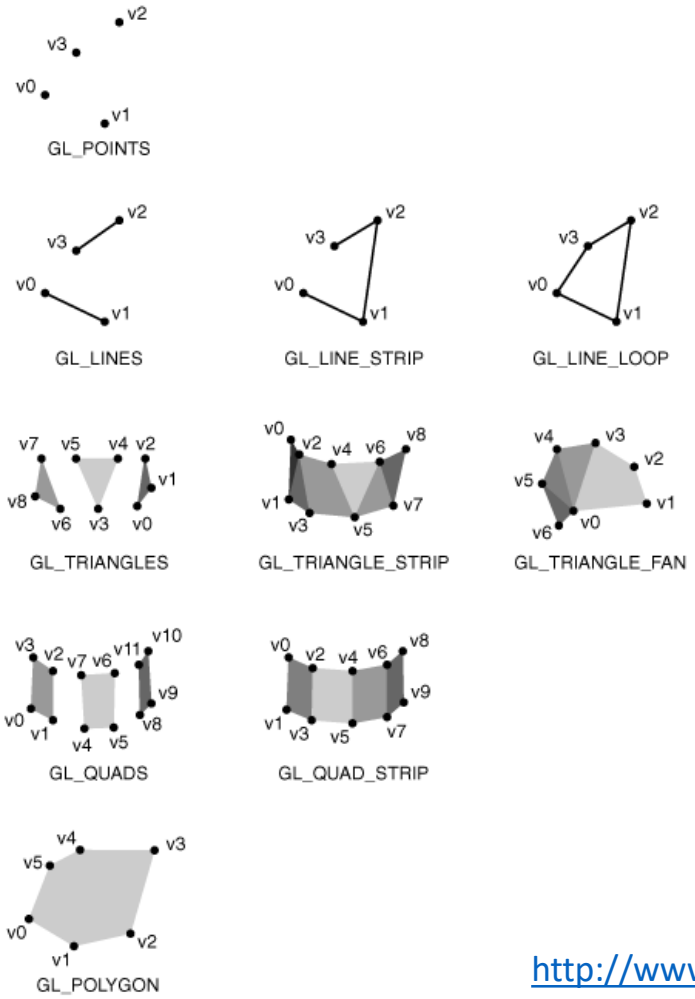
glutMotionFunc() callback to respond mouse move while a mouse button is pressed

glutPassiveMouseFunc() callback to respond to mouse motion regardless state of mouse button

glutIdleFunc() callback routine for idle state, usually used for animation

More info: http://www.opengl.org/resources/libraries/glut/spec3/node45.html

OpenGL Primitives



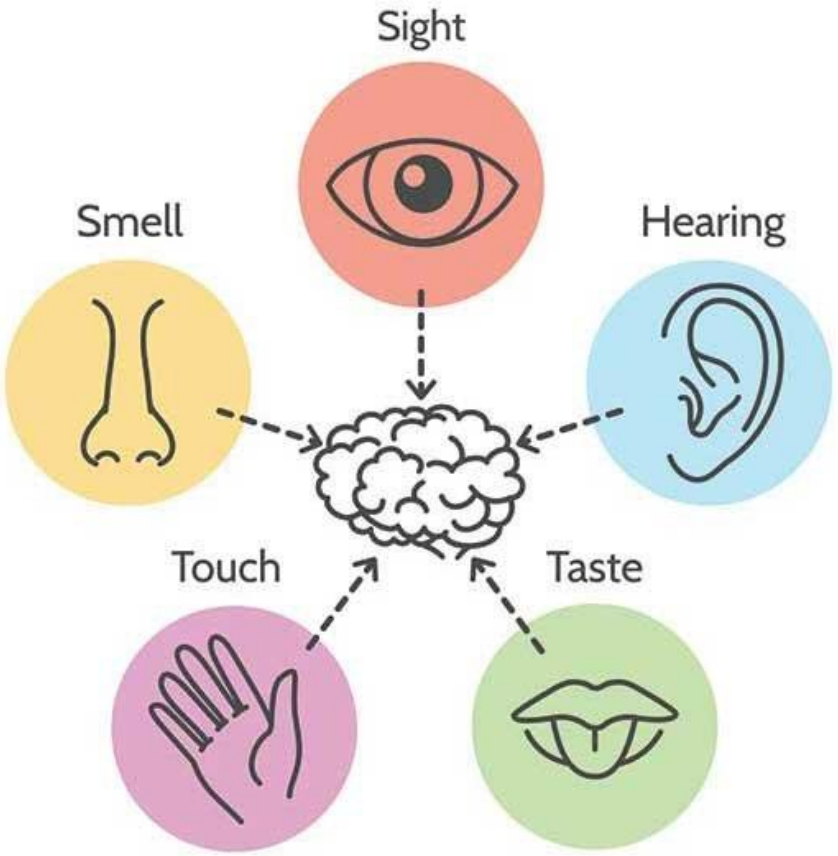

```
glBegin(GL_LINES);

  glVertex3f(10.0f, 0.0f, 0.0f);
  glVertex3f(20.0f, 0.0f, 0.0f);

  glVertex3f(10.0f, 5.0f, 0.0f);
  glVertex3f(20.0f, 5.0f, 0.0f);

glEnd();
```

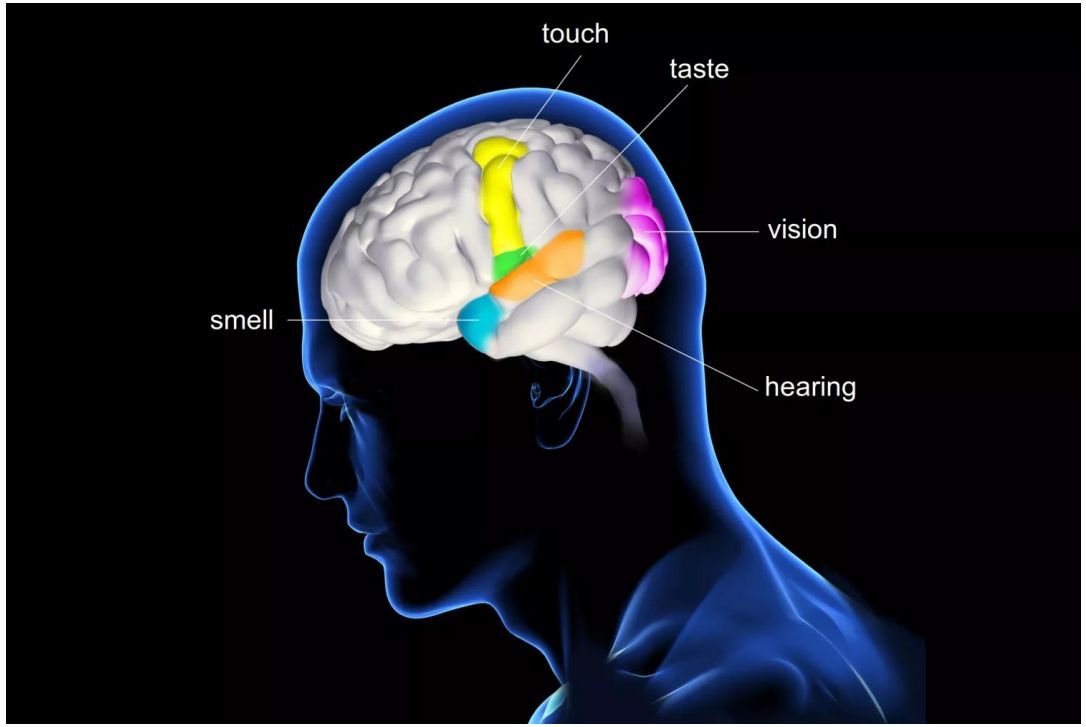
http://www.opengl.org/sdk/docs/man/xhtml/glBegin.xml

Introduction to Virtual Reality and Its Applications

Pavan Kumar B N
CSE, IIITS

How you interact with the real world?





https://www.thoughtco.com/five-senses-and-how-they-work-3888470
https://www.inspiritvr.com/general-bio/human-biology/senses-study-guide

What is Virtual Reality (VR)?

What is Virtual Reality (VR)?

Brooks (1999) defines it as: “[an] **experience** .. in which the user is effectively immersed in a responsive virtual world” ...

Sherman and Craig (2003) defines it as a **medium** composed of interactive computer simulations that sense the participant's position and actions and replace or augment the feedback to one more senses, giving the feeling of being mentally immersed or present in the simulation (a virtual world).

Virtual Reality is the use of **computer modeling and simulation** that enables a person to interact with an artificial three-dimensional visual or other sensory environment.

What is Virtual Reality?

Inducing **targeted behavior** in an **organism** by using **artificial sensory stimulation**, while the organism has little or no **awareness** of the interference.

The organism is having an "experience" that was designed by the creator

life form such as a fruit fly, cockroach, fish, rodent, or monkey

Senses of the organism become co-opted, and their inputs are enhanced by artificial stimulation

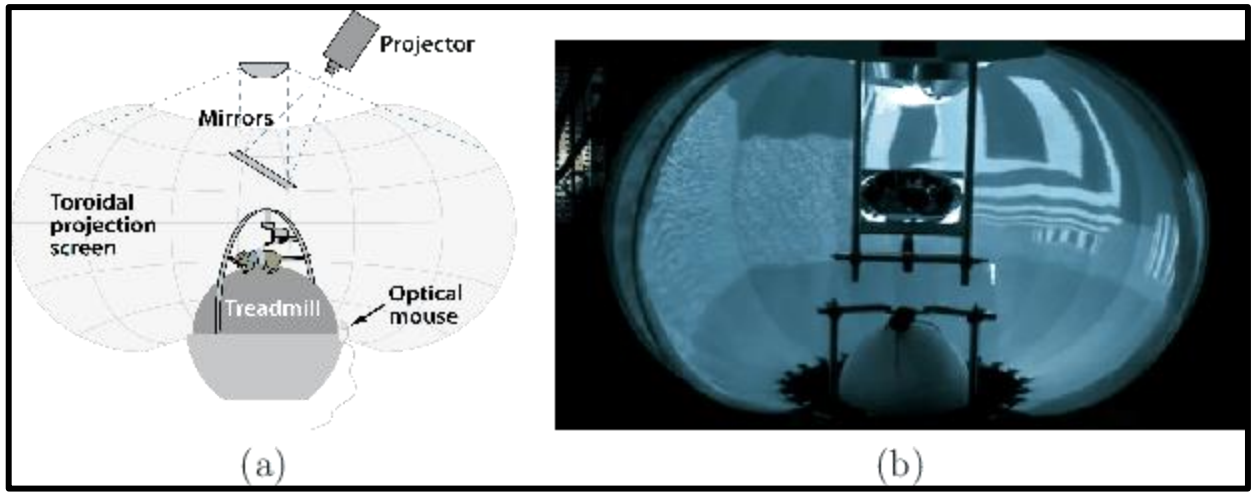
"fooled" into feeling present in a virtual world. unawareness leads to a sense of presence

http://lavalle.pl/vr/

What is Virtual Reality?

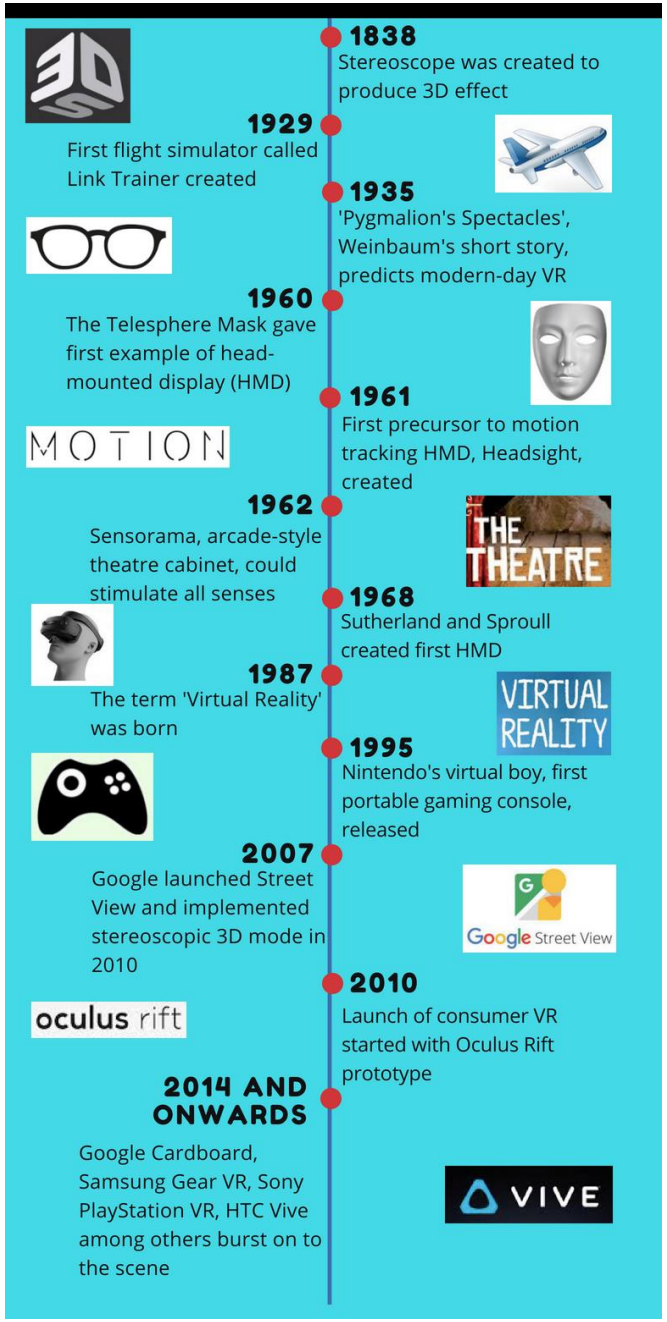
In the Birdy experience from the Zurich University of the Arts, the user, wearing a VR headset, flaps his wings while flying over virtual San Francisco

An experimental setup used by neurobiologists at LMU Munich to present visual stimuli to a gerbil while it runs on a spherical ball that acts as a treadmill



http://lavalle.pl/vr/

History

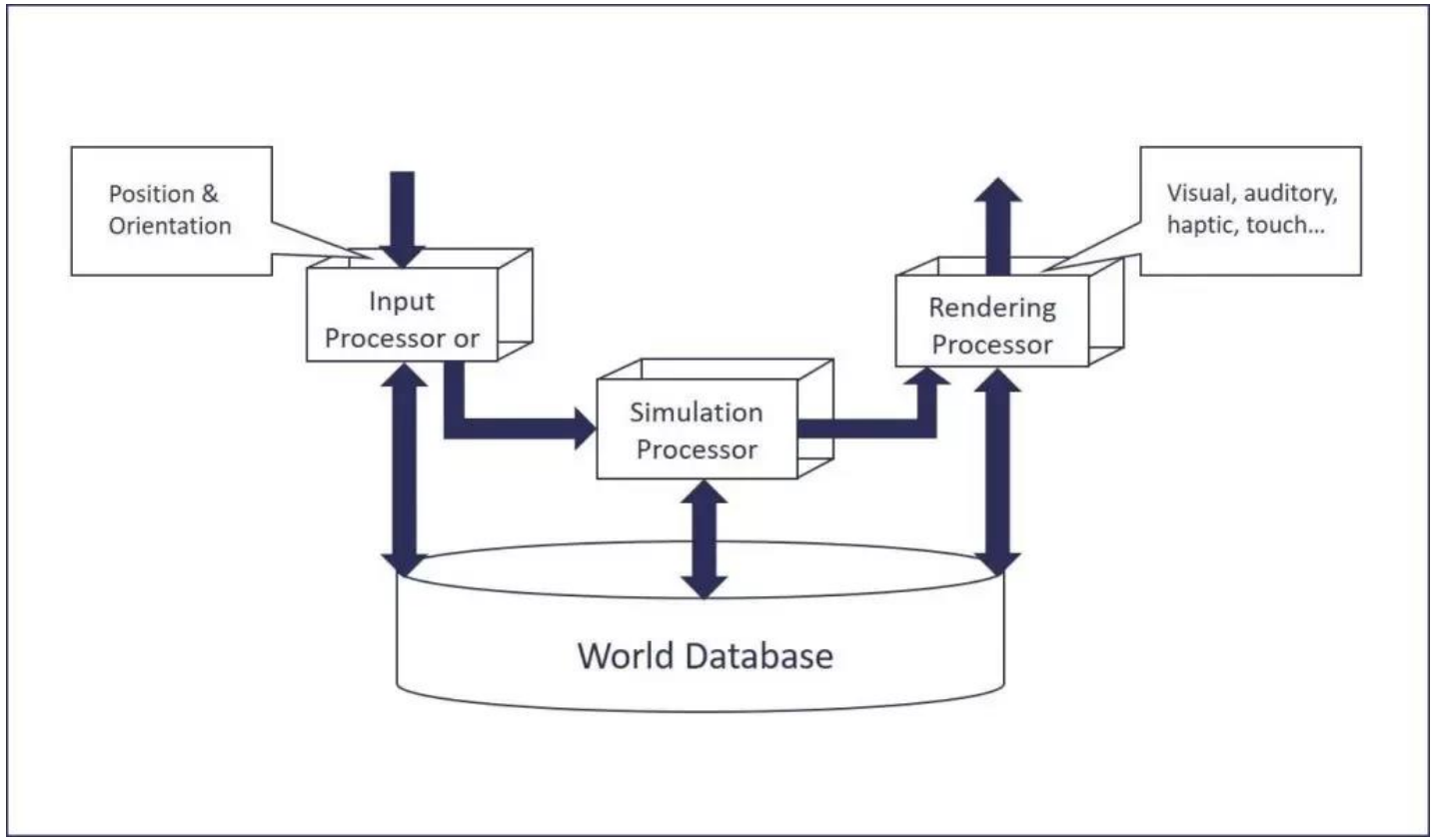


https://drupalsun.com/shankar/2018/08/24/tryst-drupal-and-virtual-reality

Key Elements

- **Immersion**: Active and Passive Immersion
- **Interactivity:** Interact with a virtual object while navigating through the environment.
- **Participants:** New user and Experienced user.
- **Feedback:** Gives ability to observe the results. Very useful in aviation and medicine.

Virtual Reality Architecture



https://www.fortech.ro/introduction-virtual-reality-world/

Virtual Reality Architecture

INPUT PROCESSOR

They’re responsible for the control of the input devices. The object is to deliver the coordinate data to the rest of the system with minimal time lag.

RENDERING PROCESSOR

Accepts the results from the simulation processor and creates the sensations that act as output to the user.

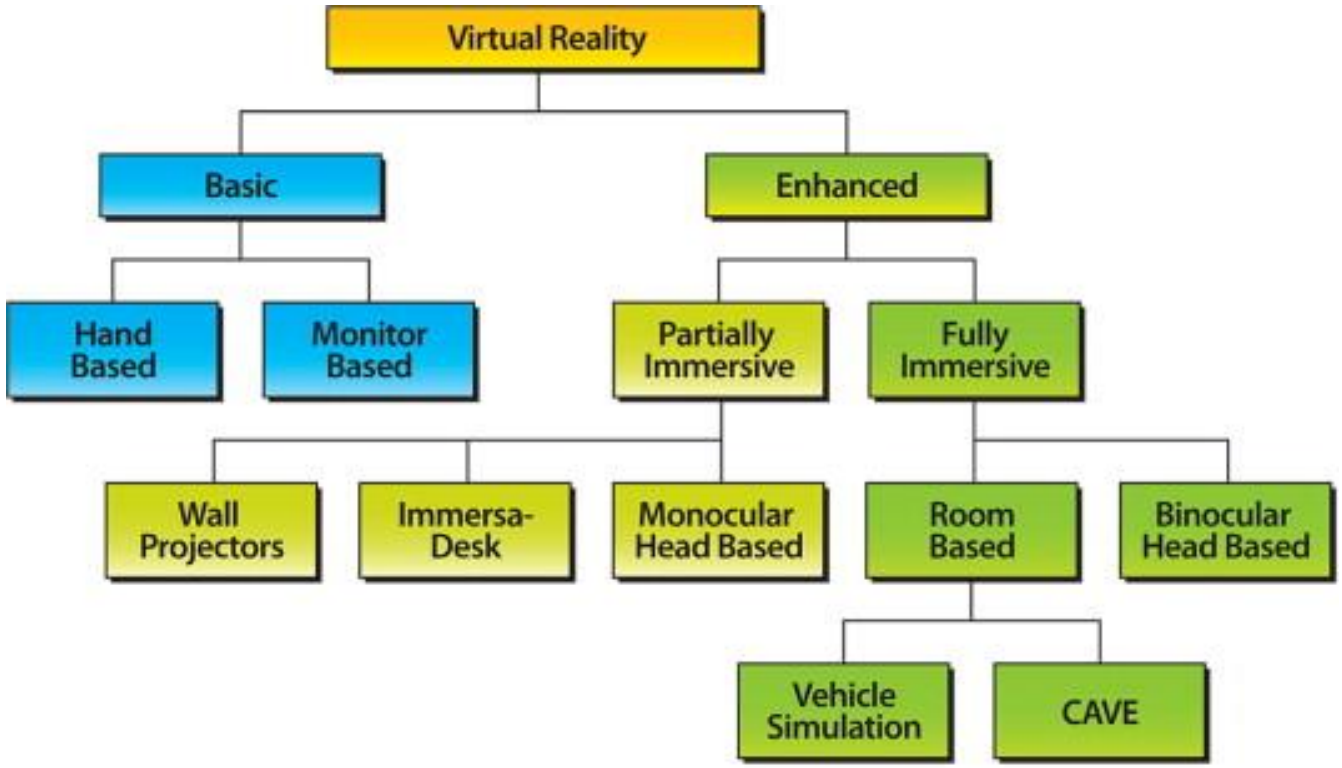
SIMULATION PROCESSOR

Core of a VR system. Takes the user inputs along with any tasks programmed and determines whether the actions that will take place in the virtual world.

WORLD DATABASE

Stores the objects that inhabit the world, scripts that describe actions of those objects.

Classification of Virtual Reality



Type of Technology:

- ***Basic:*** Do not require special input or output
- ***Enhanced:*** Require additional input or output devices to experience virtual reality.

Level of Mental Immersion:

- ***Basic :*** Lower lever of immersion
- ***Enhanced :*** Higher level of Immersion.

Muhanna, Muhanna A. "Virtual reality and the CAVE: Taxonomy, interaction challenges and research directions." *Journal of King Saud University-Computer and Information Sciences* 27, no. 3 (2015): 344-361.

Classification of Virtual Reality

Basic Systems

The basic Virtual Reality systems have the least level of immersion when compared to enhanced systems.

- These systems can be divided into subcategories, such as the **hand-based** and the **monitor-based** systems.

Leap Motion Controller

- No hand contact is required. A user can interact with their computer by just using hand gestures.
- Senses your hands and fingers and follows their every move.
- Uses 150° field of view and a Z-axis for depth, enabling a user to interact in 3D, as in real world.

Hand-based Systems

- Hand-held devices, such as cell phones, ultra mobile computers are used for Virtual Reality experience.

Monitor-based Systems

- Desktop based computers display three-dimensional graphics on monitors.
- This display provides projected stereo images from user's point of view enabling the user to see in three dimension on two dimensional monitors.

Muhanna, Muhanna A. "Virtual reality and the CAVE: Taxonomy, interaction challenges and research directions." *Journal of King Saud University-Computer and Information Sciences* 27, no. 3 (2015): 344-361.

Classification of Virtual Reality

Enhanced Systems

- Enhanced systems are again divided into two subcategories.
- **Partially Immersive :** Wall projectors, Immersa-Desk and Monocular Head Based
- **Fully Immersive:**

(a) Room based - Vehicle Simulation, CAVE

(b) Binocular head based

Muhanna, Muhanna A. "Virtual reality and the CAVE: Taxonomy, interaction challenges and research directions." *Journal of King Saud University-Computer and Information Sciences* 27, no. 3 (2015): 344-361.

How to Construct Virtual World?

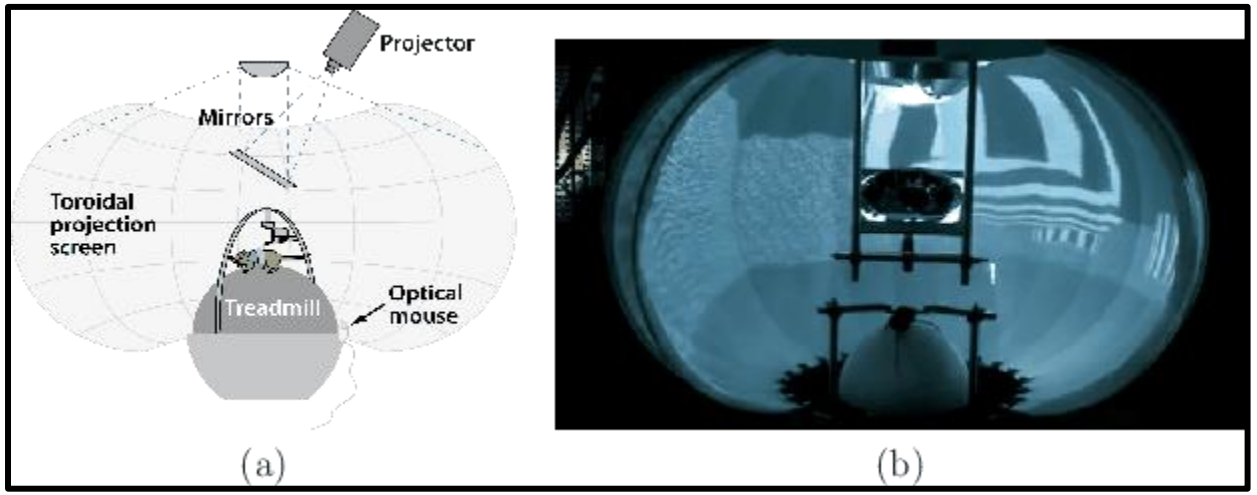
Synthetic

- We may program a *synthetic* world, which is completely invented from geometric primitives and simulated physics

Captured

- The world may be *captured* using modern imaging techniques.

- Factor affecting captured
  - What happens when the user changes her head position and viewpoint?
  - What are their facial expressions while wearing a VR headset?
  - What can we infer about their emotional state?
  - Do we need to know their hand gestures?
  - Are their eyes focused on me?



How to Construct Virtual World?

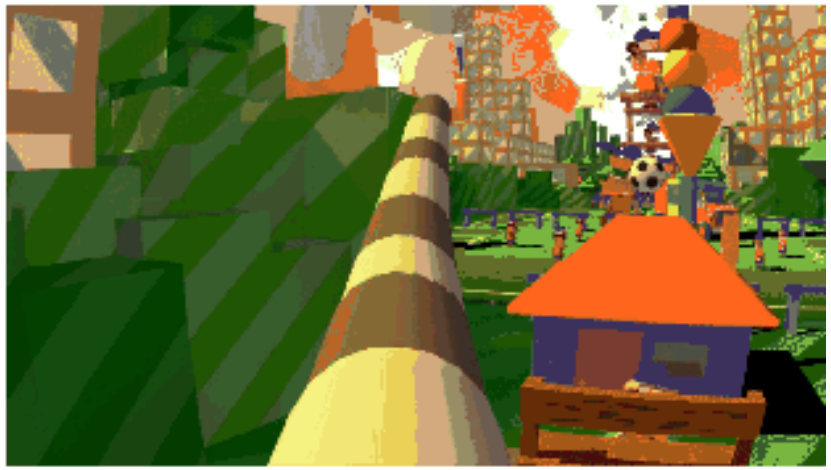
Captured

Why Virtual Reality?

- Virtual reality is adopted by military, aviation, sports like golf etc. as training ground in highly realistic manner.
- In military, soldiers can diffuse a bomb without any real-world risks.
- In Medicine
  Earlier: Dissection and study using plastic models;

  Now: Virtual Patients and Virtual Histology
- They have lower operating costs and are safer to use than real experiences.

Applications: Video Games



Immersive cinema

In VR, viewers can look in any direction, and perhaps even walk through the scene.
Some questions for the makers

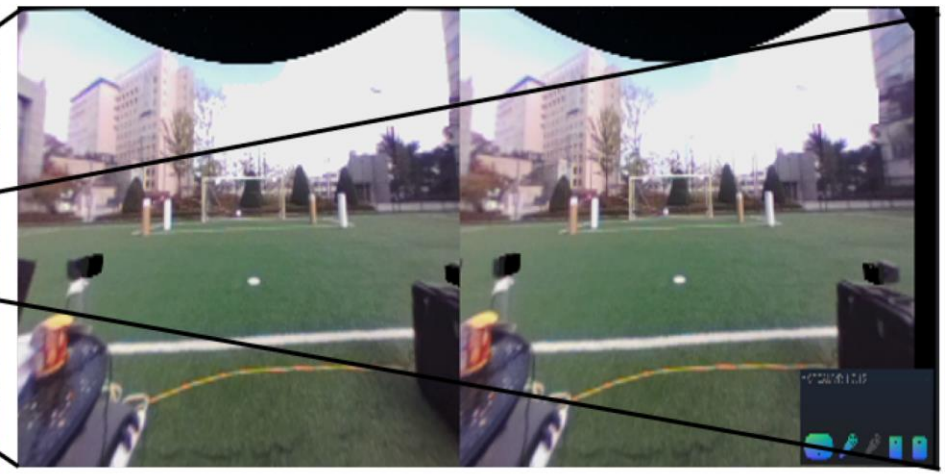
- What should they be allowed to do?
- How do you make sure they do not miss part of the story?
- Should the story be linear, or should it adapt to the viewer's actions?
- Should the viewer be a first-person character in the film, or a third-person observer who in invisible to the other characters?
- How can a group of friends experience a VR film together?
- When are animations more appropriate versus the capture of real scenes?

Telepresence

- The first step toward feeling like we are somewhere else is capturing a panoramic view of the remote environment
- Simple VR apps that query the Street View server directly enable to user to feel like he is standing in each of these locations, while easily being able to transition between nearby locations
- By connecting panoramic cameras to robots, the user is even allowed to move around in the remote environment

http://lavalle.pl/vr/

HMD View



Virtual societies

http://lavalle.pl/vr/

Empathy

In Clouds Over Sidra, 2015, offered a first-person perspective on the suffering of Syrian refugees

The Machine to Be Another

http://lavalle.pl/vr/

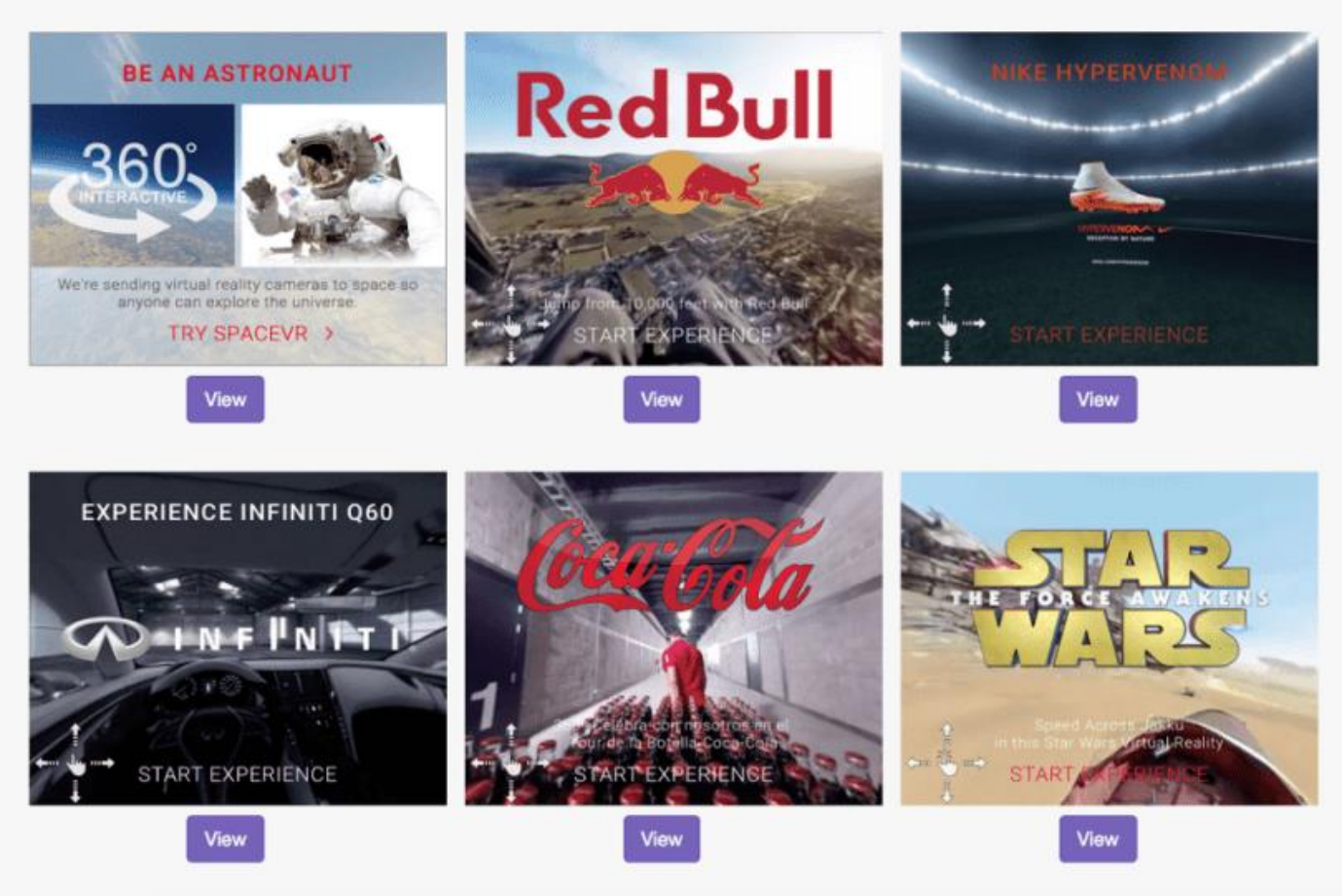
Education

A flight simulator in use by the US Air Force

A tour of the Nimrud palace

http://lavalle.pl/vr/

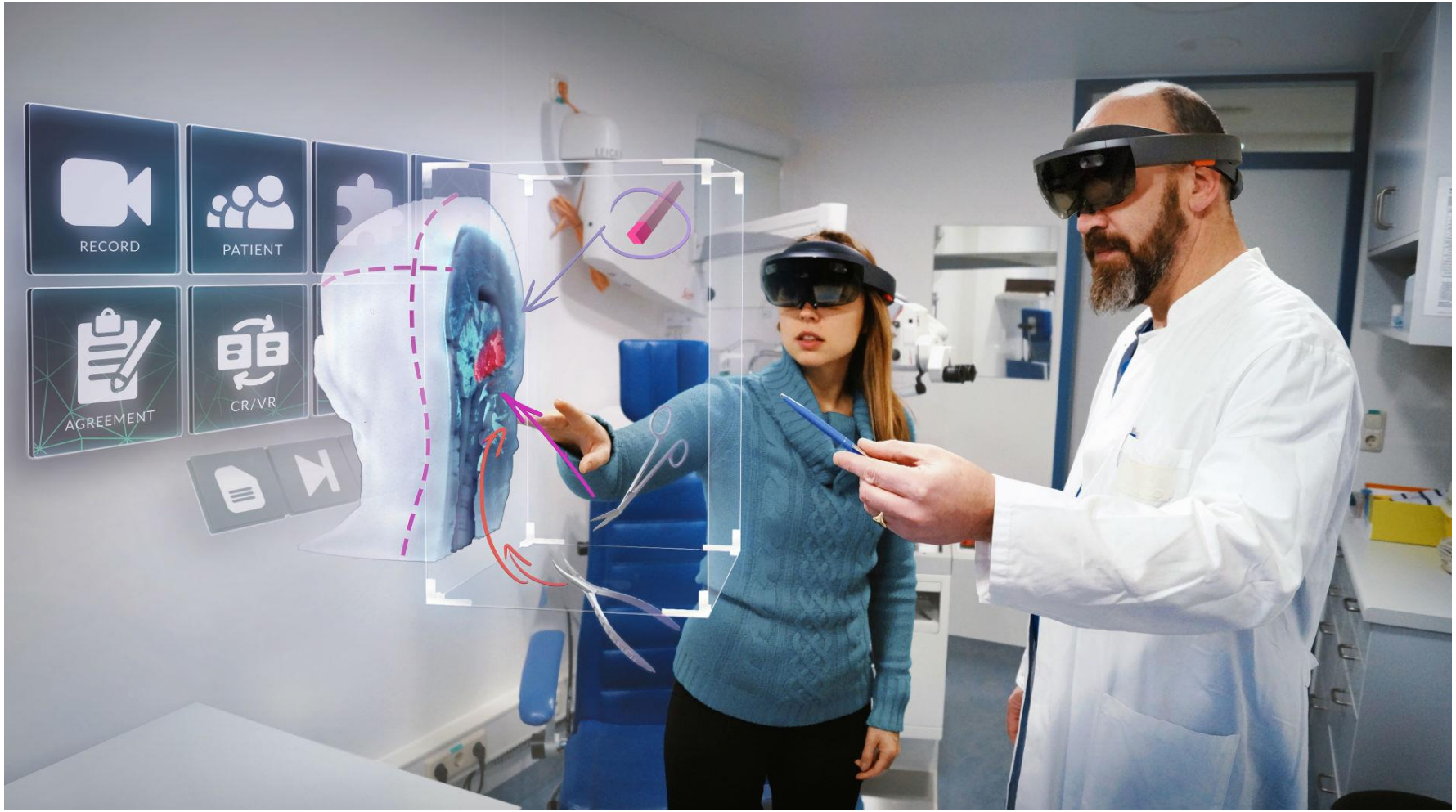
Advertisement



http://lavalle.pl/vr/

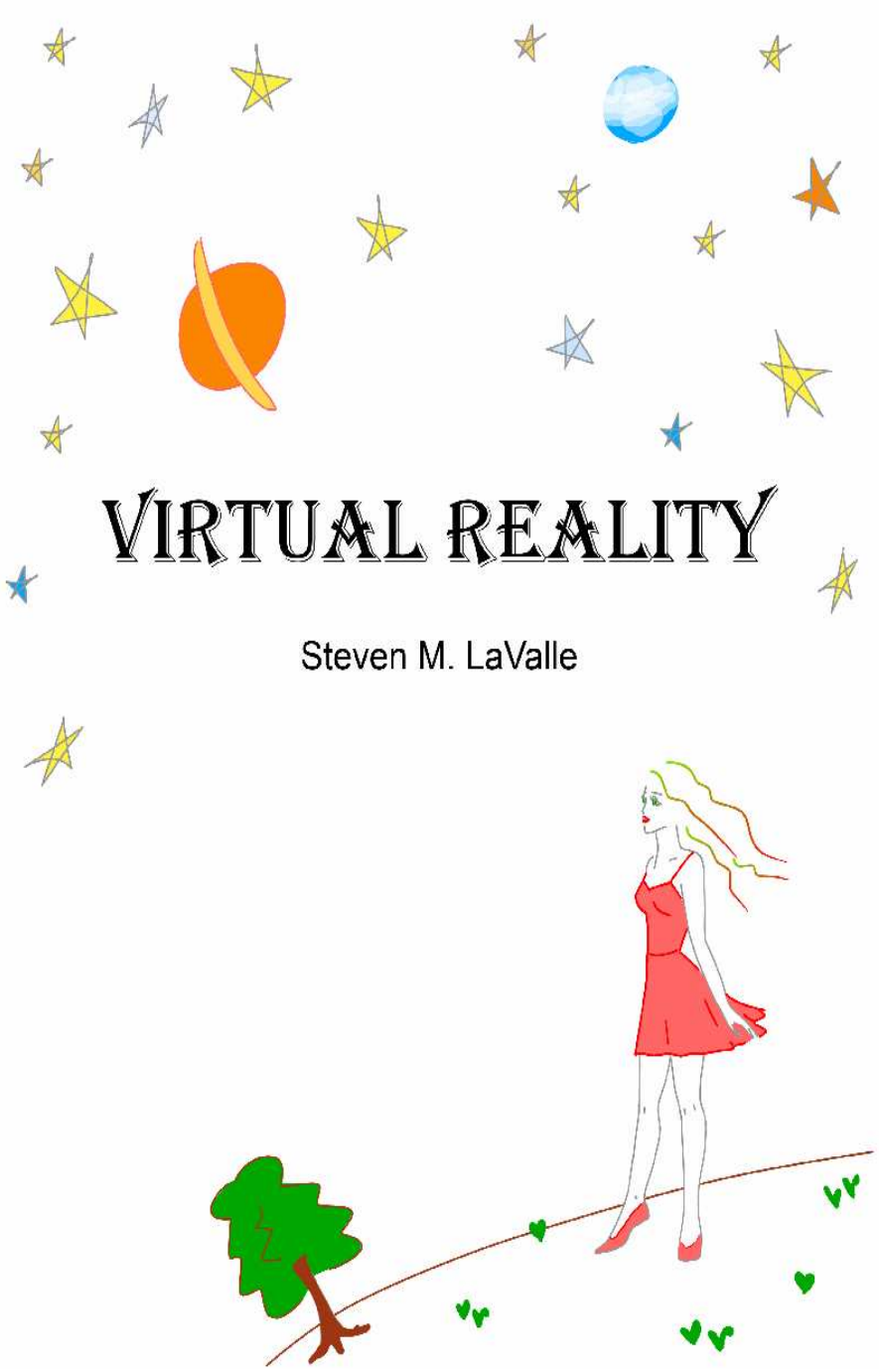
Virtual prototyping

Health care



http://lavalle.pl/vr/

THANK YOU



Chapter 1

Introduction

Steven M. LaValle

University of Oulu

Available for downloading at **http://lavalle.pl/vr/**

Chapter 1

Introduction

1.1 What Is Virtual Reality?

Virtual reality (VR) technology is evolving rapidly, making it undesirable to define VR in terms of specific devices that may fall out of favor in a year or two. In this book, we are concerned with fundamental principles that are less sensitive to particular technologies and therefore survive the test of time. Our first challenge is to consider what VR actually means, in a way that captures the most crucial aspects in spite of rapidly changing technology. The concept must also be general enough to encompass what VR is considered today and what we envision for its future.

We start with two thought-provoking examples: 1) A human having an experience of flying over virtual San Francisco by flapping his own wings (Figure 1.1); 2) a gerbil running on a freely rotating ball while exploring a virtual maze that appears on a projection screen around the mouse (Figure 1.2). We want our definition of VR to be broad enough to include these examples and many more, which are coming in Section 1.2. This motivates the following.

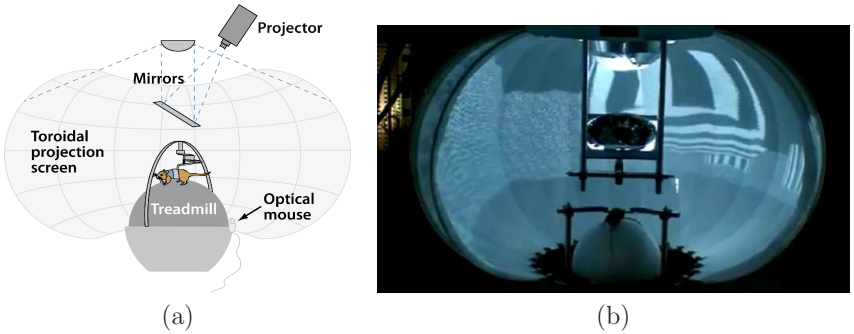
Definition of VR: Inducing targeted behavior in an organism by using artificial sensory stimulation, while the organism has little or no awareness of the interference.

Four key components appear in the definition:

1. *Targeted behavior:* The organism is having an "experience" that was designed by the creator. Examples include flying, walking, exploring, watching a movie, and socializing with other organisms.

2. *Organism:* This could be you, someone else, or even another life form such as a fruit fly, cockroach, fish, rodent, or monkey (scientists have used VR technology on all of these!).

Figure 1.1: In the Birdly experience from the Zurich University of the Arts, the user, wearing a VR headset, flaps his wings while flying over virtual San Francisco. A motion platform and fan provide additional sensory stimulation. The figure on the right shows the stimulus presented to each eye.



(a) (b)

Figure 1.2: (a) An experimental setup used by neurobiologists at LMU Munich to present visual stimuli to a gerbil while it runs on a spherical ball that acts as a treadmill (Figure from [24]). (b) A picture of a similar experiment, performed at Princeton University.

3. *Artificial sensory stimulation:* Through the power of engineering, one or more senses of the organism become co-opted, at least partly, and their ordinary inputs are replaced or enhanced by artificial stimulation.

4. *Awareness:* While having the experience, the organism seems unaware of the interference, thereby being "fooled" into feeling present in a virtual world. This unawareness leads to a sense of *presence* in an altered or alternative world. It is accepted as being natural.

You have probably seen optical illusions before. A VR system causes a *perceptual illusion* to be maintained for the organism. For this reason, human physiology and perception represent a large part of this book.

Testing the boundaries The examples shown in Figures 1.1 and 1.2 clearly fit the definition. Anyone donning a modern VR headset[1] and enjoying a session should also be included. How far does our VR definition allow one to stray from the most common examples? Perhaps listening to music through headphones should be included. What about watching a movie at a theater? Clearly, technology has been used in the form of movie projectors and audio systems to provide artificial sensory stimulation. Continuing further, what about a portrait or painting on the wall? The technology in this case involves paints and a canvass. Finally, we might even want reading a novel to be considered as VR. The technologies are writing and printing. The stimulation is visual, but does not seem as direct as a movie screen and audio system. In this book, we not worry too much about the precise boundary of our VR definition. Good arguments could be made either way about some of these borderline cases, but it is more impotant to understand the key ideas for the core of VR. The boundary cases also serve as a good point of reference for historical perspective, which is presented in Section 1.3.

Who is the fool? Returning to the VR definition above, the idea of "fooling" an organism might seem fluffy or meaningless; however, this can be made surprisingly concrete using research from neurobiology. When an animal explores its environment, neural structures composed of *place cells* are formed that encode spatial information about its surroundings [15, 16]; see Figure 1.3(a). Each place cell is activated precisely when the organism returns to a particular location that is covered by it. Although less understood, *grid cells* even encode locations in a manner similar to Cartesian coordinates [14] (Figure 1.3(b)). It has been shown that these neural structures may form in an organism, even when having a VR experience [1, 4, 9]. In other words, our brains may form place cells for places that are not real! This is a clear indication that VR is fooling our brains, at least partially. At this point, you may wonder whether reading a novel that meticulously describes an environment that does not exist will cause place cells to be generated.

[1]This is also referred to as a *head mounted display* or *HMD*.

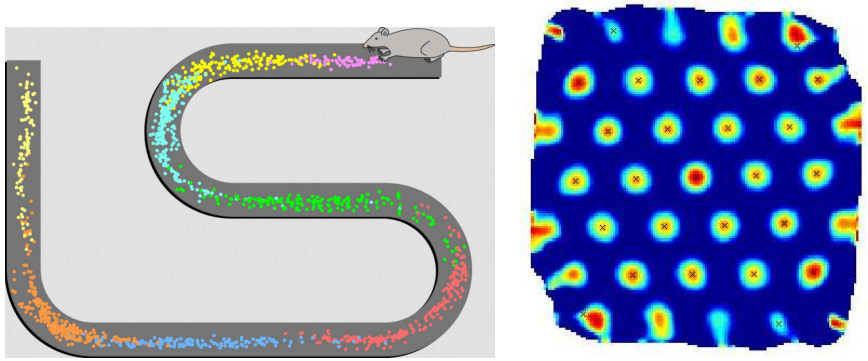


Figure 1.3: (a) We animals assign neurons as *place cells*, which fire when we return to specific locations. This figure depicts the spatial firing patterns of eight place cells in a rat brain as it runs back and forth along a winding track (figure by Stuart Layton). (b) We even have *grid cells*, which fire in uniformly, spatially distributed patterns, apparently encoding location coordinates (figure by Torkel Hafting).

Figure 1.4: A VR thought experiment: The brain in a vat, by Gilbert Harman in 1973. (Figure by Alexander Wivel.)

We also cannot help wondering whether we are always being fooled, and some greater reality has yet to reveal itself to us. This problem has intrigued the greatest philosophers over many centuries. One of the oldest instances is the *Allegory of the Cave*, presented by Plato in *Republic*. In this, Socrates describes the perspective of people who have spent their whole lives chained to a cave wall. They face a blank wall and only see shadows projected onto the walls as people pass by. He explains that the philosopher is like one of the cave people being finally freed from the cave to see the true nature of reality, rather than being only observed through projections. This idea has been repeated and popularized throughout history, and also connects deeply with spirituality and religion. In 1641, René Descartes hypothesized the idea of an *evil demon* who has directed his entire effort at deceiving humans with the illusion of the external physical world. In 1973, Gilbert Hartman introduced the idea of a *brain in a vat* (Figure 1.4), which is a thought experiment that suggests how such an evil demon might operate. This is the basis of the 1999 movie *The Matrix*. In that story, machines have fooled the entire human race by connecting to their brains to a convincing simulated world, while harvesting their real bodies. The lead character Neo must decide whether to face the new reality or take a memory-erasing pill that will allow him to comfortably live in the simulation without awareness of the ruse.

Terminology regarding various "realities" The term *virtual reality* dates back to German philosopher Immanuel Kant [25], although its use did not involve technology. Kant introduced the term to refer to the "reality" that exists in someone's mind, as differentiated from the external physical world, which is also a reality. The modern use the VR term was popularized by Jaron Lanier in the 1980s. Unfortunately, name *virtual reality* itself seems to be self contradictory, which is a philosophical problem rectified in [3] by proposing the alternative term *virtuality*. While acknowledging this issue, we will nevertheless continue onward with term *virtual reality*. The following distinction, however, will become important: The *real world* refers to the physical world that contains the user at the time of the experience, and the *virtual world* refers to the perceived world as part of the targeted VR experience.

Although the term VR is already quite encompassing, several competing terms related to VR are in common use at present. The term *virtual environments* predates widespread usage of VR and is preferred by most university researchers [8]. It is typically considered to be synonymous with VR; however, we emphasize in this book that the perceived environment could be a photographically captured "real" world just as well as a completely synthetic world. Thus, the perceived environment presented in VR need not seem "virtual". *Augmented reality* (*AR*) refers to systems in which most of the visual stimuli are propagated directly through glass or cameras to the eyes, and some additional structures, such as text and graphics, appear to be superimposed onto the user's world. The term *mixed reality* (*MR*) is sometimes used to refer to an entire spectrum that encompasses VR,

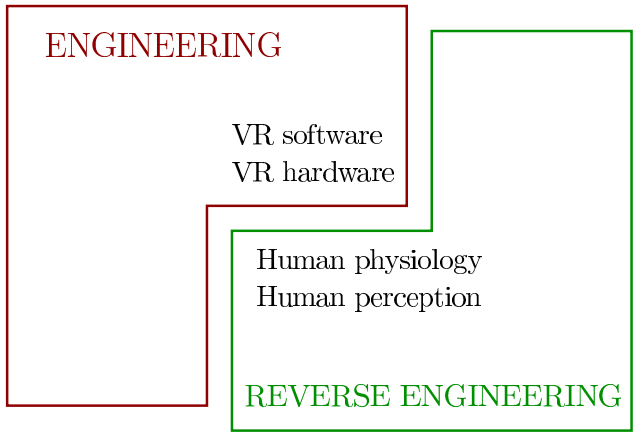


Figure 1.5: When considering a VR system, it is tempting to focus only on the traditional engineering parts: Hardware and software. However, it is equally important, if not more important, to understand and exploit the characteristics of human physiology and perception. Because we did not design ourselves, these fields can be considered as reverse engineering. All of these parts tightly fit together to form *perception engineering*.

AR, and ordinary reality [13]. People have realized that these decades-old terms and distinctions have eroded away in recent years, especially as unifying technologies have rapidly advanced. Therefore, attempts have been recently made to hastily unify them back together again under the headings *XR*, *X Reality*, *VR/AR*, *AR/VR*, *VR/AR/MR* and so on.

The related notion of *Telepresence* refers to systems that enable users to feel like they are somewhere else in the real world; if they are able to control anything, such as a flying drone, then *teleoperation* is an appropriate term. For our purposes, virtual environments, AR, mixed reality, telepresence, and teleoperation will all be considered as perfect examples of VR.

The most important idea of VR is that the user's perception of reality has been altered through engineering, rather than whether the environment they believe they are in seems more "real" or "virtual". A perceptual illusion has been engineered. Thus, another reasonable term for this area, especially if considered as an academic discipline, could be *perception engineering*, engineering methods are being used to design, develop, and deliver perceptual illusions to the user. Figure 1.5 illustrates the ingredients of perception engineering, which also motivates the topics of book, which are a mixture of engineering and human psysiology and perception.

Interactivity Most VR experiences involve another crucial component: *interaction*. Does the sensory stimulation depend on actions taken by the organism? If the answer is "no", then the VR system is called *open-loop*; otherwise, it is *closed-loop*. In the case of closed-loop VR, the organism has partial control over the sensory stimulation, which could vary as a result of body motions, including eyes, head, hands, or legs. Other possibilities include voice commands, heart rate, body temperature, and skin conductance (are you sweating?).

First- vs. Third-person Some readers of this book might want to develop VR systems or experiences. In this case, pay close attention to this next point! When a scientist designs an experiment for an organism, as shown in Figure 1.2, then the separation is clear: The laboratory subject (organism) has a *first-person* experience, while the scientist is a *third-person* observer. The scientist carefully designs the VR system as part of an experiment that will help to resolve a scientific hypothesis. For example, how does turning off a few neurons in a rat's brain affect its navigation ability? On the other hand, when engineers or developers construct a VR system or experience, they are usually targeting themselves and people like them. They feel perfectly comfortable moving back and forth between being the "scientist" and the "lab subject" while evaluating and refining their work. As you will learn throughout this book, this is a bad idea! The creators of the experience are heavily biased by their desire for it to succeed without having to redo their work. They also know what the experience is supposed to mean or accomplish, which provides a strong bias in comparison to a fresh subject. To complicate matters further, the creator's body will physically and mentally adapt to whatever flaws are present so that they may soon become invisible. You have probably seen these kinds of things before. For example, it is hard to predict how others will react to your own writing. Also, it is usually harder to proofread your own writing in comparison to that of others. In the case of VR, these effects are much stronger and yet elusive to the point that you must force yourself to pay attention to them. Take great care when hijacking the senses that you have trusted all of your life. This will most likely be uncharted territory for you.

More real than reality? How "real" should the VR experience be? It is tempting to try to make it match our physical world as closely as possible. This is referred to in Section 10.1 as the *universal simulation principle*: Any interaction mechanism in the real world can be simulated in VR. Our brains are most familiar with these settings, thereby making it seem most appropriate. This philosophy has dominated the video game industry at times, for example, in the development of highly realistic first-person shooter (FPS) games that are beautifully rendered on increasingly advanced graphics cards. In spite of this, understand that extremely simple, cartoon-like environments can also be effective and even preferable. Examples appear throughout history, as discussed in Section 1.3.

If you are a creator of VR experiences, think carefully about the task, goals,

or desired effect you want to have on the user. You have the opportunity to make the experience *better than reality*. What will they be doing? Taking a math course? Experiencing a live theatrical performance? Writing software? Designing a house? Maintaining a long-distance relationship? Playing a game? Having a meditation and relaxation session? Traveling to another place on Earth, or in the universe? For each of these, think about how the realism requirements might vary. For example, consider writing software in VR. We currently write software by typing into windows that appear on a large screen. Note that even though this is a familiar experience for many people, it was not even possible in the physical world of the 1950s. In VR, we could simulate the modern software development environment by convincing the programmer that she is sitting in front of a screen; however, this misses the point that we can create almost *anything* in VR. Perhaps a completely new interface will emerge that does not appear to be a screen sitting on a desk in an office. For example, the windows could be floating above a secluded beach or forest. Furthermore, imagine how a debugger could show the program execution trace. In all of these examples, it will important to determine the *perception-based criteria* that need to be satisfied for the perceptual illusions to be convincingly and comfortably maintained for the particular VR experience of interest.

Synthetic vs. captured Two extremes exist when constructing a virtual world as part of a VR experience. At one end, we may program a *synthetic* world, which is completely invented from geometric primitives and simulated physics. This is common in video games and such virtual environments were assumed to be the main way to experience VR in earlier decades. At the other end, the world may be *captured* using modern imaging techniques. For viewing on a screen, the video camera has served this purpose for over a century. Capturing panoramic images and videos and then seeing them from any viewpoint in a VR system is a natural extension. In many settings, however, too much information is lost when projecting the real world onto the camera sensor. What happens when the user changes her head position and viewpoint? More information should be captured in this case. Using depth sensors and SLAM (Simultaneous Localization And Mapping) techniques, a 3D representation of the surrounding world can be captured and maintained over time as it changes. It is extremely difficult, however, to construct an accurate and reliable representation, unless the environment is explicitly engineered for such capture (for example, a motion capture studio).

As humans interact, it becomes important to track their motions, which is an important form of capture. What are their facial expressions while wearing a VR headset? Do we need to know their hand gestures? What can we infer about their emotional state? Are their eyes focused on me? Synthetic representations of ourselves called *avatars* enable us to interact and provide a level of anonymity, if desired in some contexts. The attentiveness or emotional state can be generated synthetically. We can also enhance our avatars by tracking the motions and other

attributes of our actual bodies. A well-known problem is the *uncanny valley*, in which a high degree of realism has been achieved in an avatar, but its appearance makes people feel uneasy. It seems almost right, but the small differences are disturbing. There is currently no easy way to make ourselves appear to others in a VR experience exactly as we do in the real world, and in most cases, we might not want to.

Health and safety Although the degree of required realism may vary based on the tasks, one requirement remains invariant: The health and safety of the users. Unlike simpler media such as radio or television, VR has the power to overwhelm the senses and the brain, leading to fatigue or sickness. This phenomenon has been studied under the heading of *simulator sickness* for decades; in this book we will refer to adverse symptoms from VR usage as *VR sickness*. Sometimes the discomfort is due to problems in the VR hardware and low-level software; however, in many cases, it is caused by a careless developer who misunderstands or disregards the side effects of the experience on the user. This is one reason why human physiology and perceptual psychology are large components of this book. To engineer comfortable VR experiences, one must understand how these factor in. In many cases, fatigue arises because the brain appears to work harder to integrate the unusual stimuli being presented to the senses. In some cases, inconsistencies with prior expectations, and outputs from other senses, even lead to dizziness and nausea.

Another factor that leads to fatigue is an interface that requires large amounts of muscular effort. For example, it might be tempting move objects around in a sandbox game by moving your arms around in space. This quickly leads to fatigue and an avoidable phenomenon called *gorilla arms*, in which people feel that the weight of their extended arms is unbearable. For example, by following the principle of the computer mouse, it may be possible to execute large, effective motions in the virtual world by small, comfortable motions of a controller. Over long periods of time, the brain will associate the motions well enough for it to seem realistic while also greatly reducing fatigue. This will be revisited in Section ??.

1.2 Modern VR Experiences

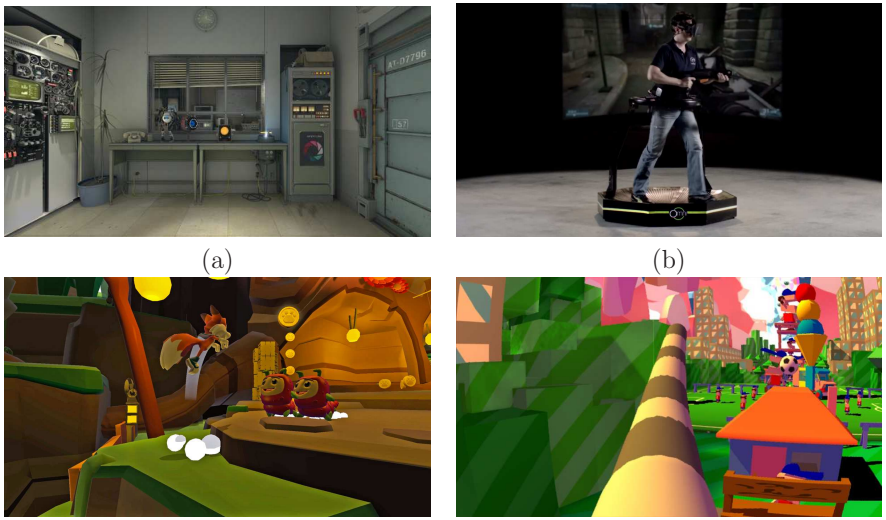
The current generation of VR systems was brought about by advances in display, sensing, and computing technology from the smartphone industry. From Palmer Luckey's 2012 Oculus Rift design to building a viewing case for smart phones [10, 17, 20], the world has quickly changed as VR headsets are mass produced and placed onto the heads of millions of people. This trend is similar in many ways to the home computer and web browser revolutions; as a wider variety of people have access to the technology, the set of things they do with it substantially broadens.

This section provides a quick overview of what people are doing with VR systems, and provides a starting point for searching for similar experiences on the Internet. Here, we can only describe the experiences in words and pictures, which is a long way from the appreciation gained by experiencing them yourself. This printed medium (a book) is woefully inadequate for fully conveying the medium of VR. Perhaps this is how it was in the 1890s to explain in a newspaper what a movie theater was like! If possible, it is strongly recommended that you try many VR experiences yourself to form first-hand opinions and spark your imagination to do something better.

Video games People have dreamed of entering their video game worlds for decades. By 1982, this concept was already popularized by the Disney movie Tron. Figure 1.6 shows several video game experiences in VR. Most gamers currently want to explore large, realistic worlds through an avatar. Figure 1.6(a) shows Valve's Portal 2 for the HTC Vive headset. Figure 1.6(b) shows an omnidirectional treadmill peripheral that gives users the sense of walking while they slide their feet in a dish on the floor. These two examples give the user a *first-person* perspective of their character. By contrast, Figure 1.6(c) shows Lucky's Tale, which instead yields a comfortable *third-person* perspective as the user seems to float above the character that she controls. Figure 1.6(d) shows a game that contrasts all the others in that it was designed to specifically exploit the power of VR.

Immersive cinema Hollywood movies continue to offer increasing degrees of realism. Why not make the viewers feel like they are part of the scene? Figure 1.7 shows an immersive short story. Movie directors are entering a fascinating new era of film. The tricks of the trade that were learned across the 20th century need to be reinvestigated because they are based on the assumption that the cinematographer controls the camera viewpoint. In VR, viewers can look in any direction, and perhaps even walk through the scene. What should they be allowed to do? How do you make sure they do not miss part of the story? Should the story be linear, or should it adapt to the viewer's actions? Should the viewer be a first-person character in the film, or a third-person observer who in invisible to the other characters? How can a group of friends experience a VR film together? When are animations more appropriate versus the capture of real scenes?

It will take many years to resolve these questions and countless more that will arise. In the meantime, VR can also be used as a kind of "wrapper" around existing movies. Figure 1.8 shows the VR Cinema application, which allows the user to choose any seat in a virtual movie theater. Whatever standard movies or videos that are on the user's hard drive can be streamed to the screen in the theater. These could be 2D or 3D movies. A projector in the back emits flickering lights and the audio is adjusted to mimic the acoustics of a real theater. This provides an immediate way to leverage all content that was developed for viewing on a screen, and bring it into VR. Many simple extensions can be made



(a) (b) (c) (d)

Figure 1.6: (a) Valve's Portal 2 demo, which shipped with The Lab for the HTC Vive headset, is a puzzle-solving experience in a virtual world. (b) The Virtuix Omni treadmill for walking through first-person shooter games. (c) Lucky's Tale for the Oculus Rift maintains a third-person perspective as the player floats above his character. (d) In the Dumpy game from DePaul University, the player appears to have a large elephant trunk. The purpose of the game is to enjoy this unusual embodiment by knocking things down with a swinging trunk.

without modifying the films. For example, in a movie about zombies, a few virtual zombies could enter the theater and start to chase you. In a movie about tornadoes, perhaps the theater rips apart. You can also have a social experience. Imagine having "movie night" with your friends from around the world, while you sit together in the virtual movie theater. You can even have the thrill of misbehaving in the theater without getting thrown out.

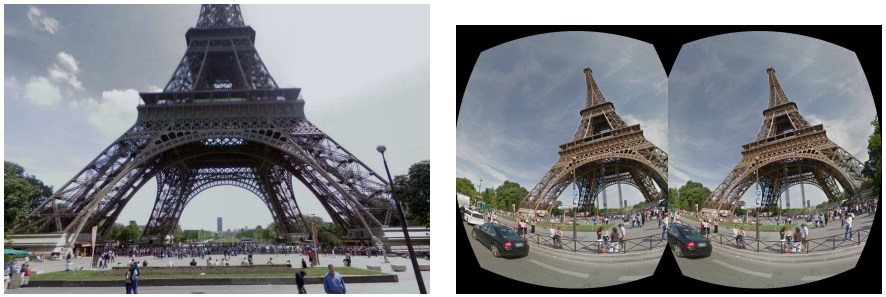
Telepresence The first step toward feeling like we are somewhere else is capturing a panoramic view of the remote environment (Figure 1.9). Google's Street View and Earth apps already rely on the captured panoramic images from millions of locations around the world. Simple VR apps that query the Street View server directly enable to user to feel like he is standing in each of these locations, while easily being able to transition between nearby locations (Figure 1.10). Panoramic video capture is even more compelling. Figure 1.11 shows a frame from an immersive rock concert experience. Even better is to provide *live* panoramic video interfaces, through which people can attend sporting events and concerts. Through

Figure 1.7: In 2015, Oculus Story Studio produced Emmy-winning *Henry*, an immersive short story about an unloved hedgehog who hopes to make a new friend, the viewer.

Figure 1.8: VR Cinema, developed in 2013 by Joo-Hyung Ahn for the Oculus Rift. Viewers could choose their seats in the theater and watch any movie they like.

(a) (b)

Figure 1.9: An important component for achieving telepresence is to capture a panoramic view: (a) A car with cameras and depth sensors on top, used by Google to make Street View. (b) The Insta360 Pro captures and streams omnidirectional videos.



(a) (b)

Figure 1.10: A simple VR experience that presents Google Street View images through a VR headset: (a) A familiar scene in Paris. (b) Left and right eye views are created inside the headset, while also taking into account the user's looking direction.

Figure 1.11: Jaunt captured a panoramic video of Paul McCartney performing Live and Let Die, which provides a VR experience where users felt like they were on stage with the rock star.

(a) (b)

Figure 1.12: Examples of robotic avatars: (a) The DORA robot from the University of Pennsylvania mimics the users head motions, allowing him to look around in a remote world while maintaining a stereo view (panoramas are monoscopic). (b) The Plexidrone, a flying robot that is designed for streaming panoramic video.

a live interface, interaction is possible. People can take video conferencing to the next level by feeling present at the remote location. By connecting panoramic cameras to robots, the user is even allowed to move around in the remote environment (Figure 1.12). Current VR technology allows us to virtually visit far away places and interact in most of the ways that were previously possible only while physically present. This leads to improved opportunities for telecommuting to work. This could ultimately help reverse the urbanization trend sparked by the 19th-century industrial revolution, leading to *deurbanization* as we distribute more uniformly around the Earth.

Virtual societies Whereas telepresence makes us feel like we are in another part of the physical world, VR also allows us to form entire societies that remind us of the physical world, but are synthetic worlds that contain avatars connected to real people. Figure 1.13 shows a Second Life scene in which people interact in a fantasy world through avatars; such experiences were originally designed to view on a screen but can now be experienced through VR. Groups of people could spend time together in these spaces for a variety of reasons, including common special interests, educational goals, or simply an escape from ordinary life.

Empathy The first-person perspective provided by VR is a powerful tool for causing people to feel *empathy* for someone else's situation. The world continues to struggle with acceptance and equality for others of different race, religion, age, gender, sexuality, social status, and education, while the greatest barrier to progress is that most people cannot fathom what it is like to have a different identity. Figure 1.14 shows a VR project sponsored by the United Nations to

Figure 1.13: Virtual societies develop through interacting avatars that meet in virtual worlds that are maintained on a common server. A snapshot from Second Life is shown here.

Figure 1.14: In Clouds Over Sidra, 2015, film producer Chris Milk offered a first-person perspective on the suffering of Syrian refugees (figure by Within, Clouds Over Sidra).

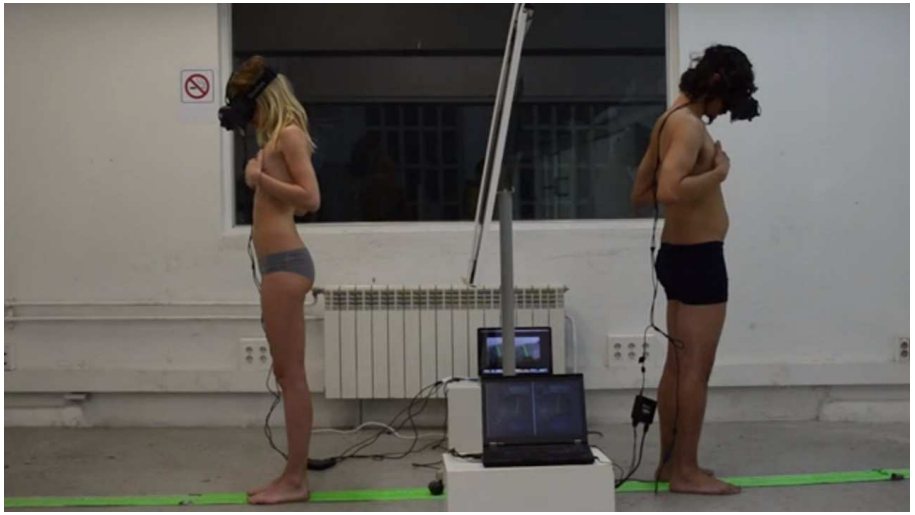


Figure 1.15: In 2014, BeAnotherLab, an interdisciplinary collective, made "The Machine to Be Another" where you can swap bodies with the other gender. Each person wears a VR headset that has cameras mounted on its front. Each therefore sees the world from the approximate viewpoint of the other person. They were asked to move their hands in coordinated motions so that they see their new body moving appropriately.

Figure 1.16: A flight simulator in use by the US Air Force (photo by Javier Garcia, U.S. Air Force). The user sits in a physical cockpit while being surrounded by displays that show the environment.

yield feelings of empathy for those caught up in the Syrian crisis of 2015. Some of us may have compassion for the plight of others, but it is a much stronger feeling to understand their struggle because you have been there before. Figure 1.15 shows a VR system that allows men and women to swap bodies. Through virtual societies, many more possibilities can be explored. What if you were 10cm shorter than everyone else? What if you teach your course with a different gender? What if you were the victim of racial discrimination by the police? Using VR, we can imagine many "games of life" where you might not get as far without being in the "proper" group.

Education In addition to teaching empathy, the first-person perspective could revolutionize many areas of education. In engineering, mathematics, and the sciences, VR offers the chance to visualize geometric relationships in difficult concepts or data that are hard to interpret. Furthermore, VR is naturally suited for practical training because skills developed in a realistic virtual environment may transfer naturally to the real environment. The motivation is particularly high if the real environment is costly to provide or poses health risks. One of the earliest and most common examples of training in VR is *flight simulation* (Figure 1.16).

Figure 1.17: A tour of the Nimrud palace of Assyrian King Ashurnasirpal II, a VR experience developed by Learning Sites Inc. and the University of Illinois in 2016.

Other examples include firefighting, nuclear power plant safety, search-and-rescue, military operations, and medical procedures.

Beyond these common uses of VR, perhaps the greatest opportunities for VR education lie in the humanities, including history, anthropology, and foreign language acquisition. Consider the difference between reading a book on the Victorian era in England and being able to roam the streets of 19th-century London, in a simulation that has been painstakingly constructed by historians. We could even visit an ancient city that has been reconstructed from ruins (Figure 1.17). Fascinating possibilities exist for either touring *physical* museums through a VR interface or scanning and exhibiting artifacts directly in *virtual* museums. These examples fall under the heading of *digital heritage*.

Virtual prototyping In the real world, we build prototypes to understand how a proposed design feels or functions. Thanks to 3D printing and related technologies, this is easier than ever. At the same time, *virtual prototyping* enables designers to inhabit a virtual world that contains their prototype (Figure 1.18). They can quickly interact with it and make modifications. They also have opportunities to bring clients into their virtual world so that they can communicate their ideas. Imagine you want to remodel your kitchen. You could construct a model in VR and then explain to a contractor exactly how it should look. Virtual prototyping in VR has important uses in many businesses, including real estate, architecture, and the design of aircraft, spacecraft, cars, furniture, clothing, and

Figure 1.18: Architecture is a prime example of where a virtual prototype is invaluable. This demo, called Ty Hedfan, was created by IVR-NATION. The real kitchen is above and the virtual kitchen is below.

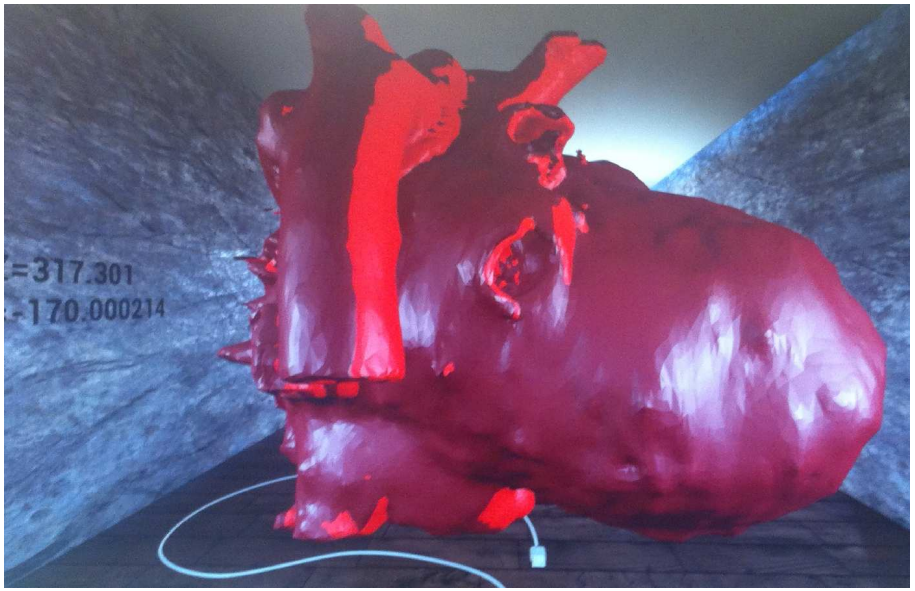


Figure 1.19: A heart visualization system based on images of a real human heart. This was developed by the Jump Trading Simulation and Education Center and the University of Illinois.

medical instruments.

Health care Although health and safety are challenging VR issues, the technology can also help to improve our health. There is an increasing trend toward distributed medicine, in which doctors train people to perform routine medical procedures in remote communities around the world. Doctors can provide guidance through telepresence, and also use VR technology for training. In another use of VR, doctors can immerse themselves in 3D organ models that were generated from medical scan data (Figure 1.19). This enables them to better plan and prepare for a medical procedure by studying the patient's body shortly before an operation. They can also explain medical options to the patient or his family so that they may make more informed decisions. In yet another use, VR can directly provide therapy to help patients. Examples include overcoming phobias and stress disorders through repeated exposure, improving or maintaining cognitive skills in spite of aging, and improving motor skills to overcome balance, muscular, or nervous system disorders. VR systems could also one day improve longevity by enabling aging people to virtually travel, engage in fun physical therapy, and overcome loneliness by connecting with family and friends through an interface that makes them feel present and included in remote activities.

Figure 1.20: The Microsoft Hololens, 2016, uses advanced see-through display technology to superimpose graphical images onto the ordinary physical world, as perceived by looking through the glasses.

Augmented and mixed reality In many applications, it is advantageous for users to see the live, real world with some additional graphics superimposed to enhance its appearance; see Figure 1.20. This has been referred to as *augmented reality* or *mixed reality* (both of which we consider to be part of VR in this book). By placing text, icons, and other graphics into the real world, the user could leverage the power of the Internet to help with many operations such as navigation, social interaction, and mechanical maintenance. Many applications to date are targeted at helping businesses to conduct operations more efficiently. Imagine a factory environment in which workers see identifying labels above parts that need to assembled, or they can look directly inside of a machine to determine potential replacement parts.

These applications rely heavily on advanced computer vision techniques, which must identify objects, reconstruct shapes, and identify lighting sources in the real world before determining how to draw virtual objects that appear to be naturally embedded. Achieving a high degree of reliability becomes a challenge because vision algorithms make frequent errors in unforeseen environments. The real-world lighting conditions must be estimated to determine how to draw the virtual objects and any shadows they might cast onto real parts of the environment and other virtual objects. Furthermore, the real and virtual objects may need to be perfectly aligned in some use cases, which places strong burdens on both tracking and computer vision systems.

Several possibilities exist for visual displays. A fixed screen should show images

Figure 1.21: Nintendo Pokemon Go is a geolocation-based game from 2016 that allows users to imagine a virtual world that is superimposed on to the real world. They can see Pokemon characters only by looking "through" their smartphone screen.

(a) (b)

Figure 1.22: (a) In 2014, Epic Games created a wild roller coaster ride through virtual living room. (b) A guillotine simulator was made in 2013 by Andre Berlemont, Morten Brunbjerg, and Erkki Trummal. Participants were hit on the neck by friends as the blade dropped, and they could see the proper perspective as their heads rolled.

that are enhanced through 3D glasses. A digital projector could augment the environment by shining light onto objects, giving them new colors and textures, or by placing text into the real world. A handheld screen, which is part of a smartphone or tablet could be used as a window into the augmented or mixed world. This is the basis of the popular Nintendo Pokemon Go game; Figure 1.21. The cases more relevant for this book involve mounting the display on the head. In this case, two main approaches exist. In a *see-through display*,, the users see most of the real world by simply looking through a transparent material, while the virtual objects appear on the display to disrupt part of the view. Recent prototype headsets with advanced see-through display technology include Google Glass, Microsoft Hololens, and Magic Leap. Achieving high resolution, wide field of view, and the ability to block out incoming light remain significant challenges for affordable consumer-grade devices; however, it may become well-solved within a few years. An alternative is a *pass-through display*, which sends images from an outward-facing camera to a standard screen inside of the headset. Pass-through displays overcome current see-through display problems, but instead suffer from latency, optical distortion, color distortion, and limited dynamic range.

New human experiences Finally, the point might be to simply provide a new human experience. Through telepresence, people can try experiences through the eyes of robots or other people. However, we can go further by giving people experiences that are impossible (or perhaps deadly) in the real world. Most often, artists are the ones leading this effort. The Birdly experience of human flying

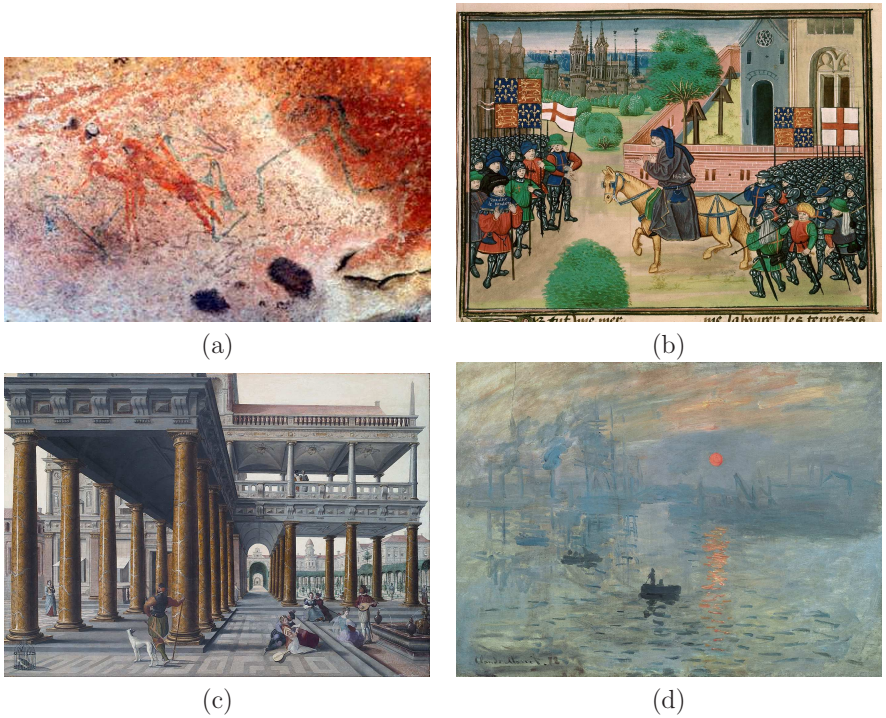
(Figure 1.1) was an excellent example. Figure 1.22 shows two more. What if we change our scale? Imagine being 2mm tall and looking ants right in the face. Compare that to being 50m tall and towering over a city while people scream and run from you. What if we simulate the effect of drugs in your system? What if you could become your favorite animal? What if you became a piece of food? The creative possibilities for artists seem to be endless. We are limited only by what our bodies can comfortably handle. Exciting adventures lie ahead!

1.3 History Repeats

Staring at rectangles How did we arrive to VR as it exists today? We start with a history that predates what most people would consider to be VR, but includes many aspects crucial to VR that have been among us for tens of thousands of years. Long ago, our ancestors were trained to look at the walls and imagine a 3D world that is part of a story. Figure 1.23 shows some examples of this. Cave paintings, such as the one shown in Figure 1.23(a) from 30,000 years ago. Figure 1.23(b) shows a painting from the European Middle Ages. Similar to the cave painting, it relates to military conflict, a fascination of humans regardless of the era or technology. There is much greater detail in the newer painting, leaving less to the imagination; however, the drawing perspective is comically wrong. Some people seem short relative to others, rather than being further away. The rear portion of the fence looks incorrect. Figure 1.23(c) shows a later painting in which the perspective has been meticulously accounted for, leading to a beautiful palace view that requires no imagination for us to perceive it as "3D". By the 19th century, many artists had grown tired of such realism and started the controversial *impressionist* movement, an example of which is shown in Figure 1.23(d). Such paintings leave more to the imagination of the viewer, much like the earlier cave paintings.

Moving pictures Once humans were content with staring at rectangles on the wall, the next step was to put them into motion. The phenomenon of *stroboscopic apparent motion* is the basis for what we call *movies* or *motion pictures* today. Flipping quickly through a sequence of pictures gives the illusion of motion, even at a rate as low as two pictures per second. Above ten pictures per second, the motion even appears to be continuous, rather than perceived as individual pictures. One of the earliest examples of this effect is the race horse movie created by Eadward Muybridge in 1878 at the request of Leland Stanford (yes, that one!); see Figure 1.24.

Motion picture technology quickly improved, and by 1896, a room full of spectators in a movie theater screamed in terror as a short film of a train pulling into a station convinced them that the train was about to crash into them (Figure 1.25(a)). There was no audio track. Such a reaction seems ridiculous for anyone who has been to a modern movie theater. As audience expectations increased,



(a) (b) (c) (d)

Figure 1.23: (a) A 30,000-year-old painting from the Bhimbetka rock shelters in India (photo by Archaelogical Survey of India). (b) An English painting from around 1470 that depicts John Ball encouraging Wat Tyler rebels (unknown artist). (c) A painting by Hans Vredeman de Vries in 1596. (d) An impressionist painting by Claude Monet in 1874.

so had the degree of realism produced by special effects. In 1902, viewers were inspired by a Journey to the Moon (Figure 1.25(b)), but by 2013, an extremely high degree of realism seemed necessary to keep viewers believing (Figure 1.25(c) and 1.25(d)).

At the same time, motion picture audiences have been willing to accept lower degrees of realism. One motivation, as for paintings, is to leave more to the imagination. The popularity of *animation* (also called *anime* or *cartoons*) is a prime example (Figure 1.26). Even within the realm of animations, a similar trend has emerged as with motion pictures in general. Starting from simple line drawings in 1908 with Fantasmagorie (Figure 1.26(a)), greater detail appears in 1928 with the introduction of Mickey Mouse(Figure 1.26(b)). By 2003, animated films achieved a much higher degree of realism (Figure 1.26(c)); however, excessively simple animations have also enjoyed widespread popularity (Figure 1.26(d)).

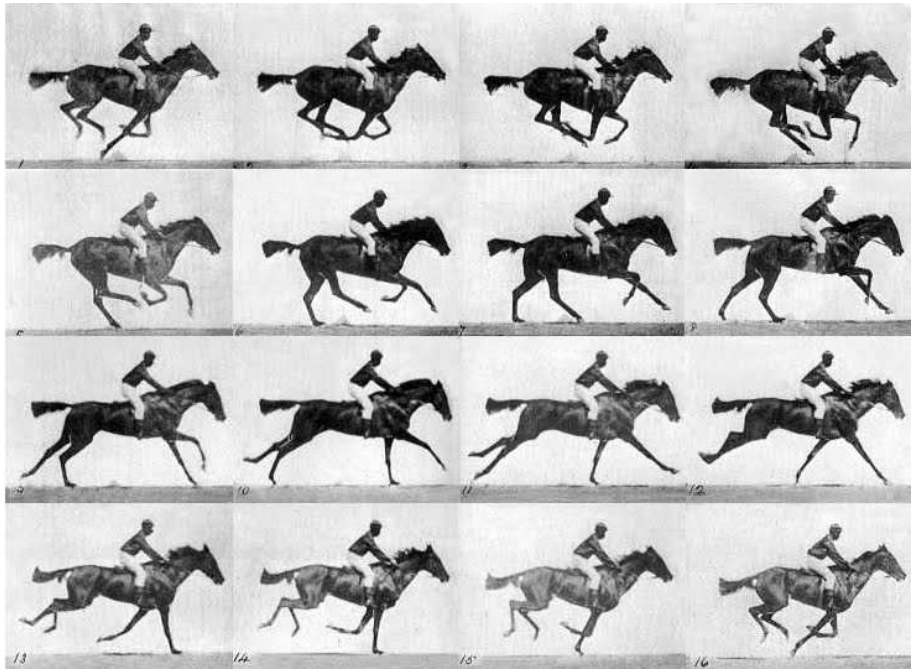
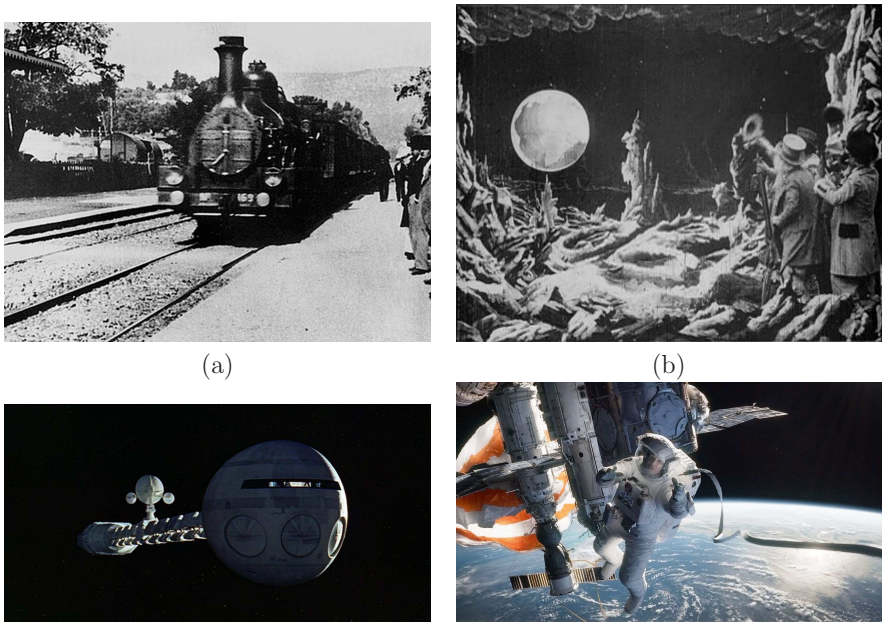


Figure 1.24: This 1878 *Horse in Motion* motion picture by Eadward Muybridge, was created by evenly spacing 24 cameras along a track and triggering them by trip wire as the horse passes. The animation was played on a zoopraxiscope, which was a precursor to the movie projector, but was mechanically similar to a record player.

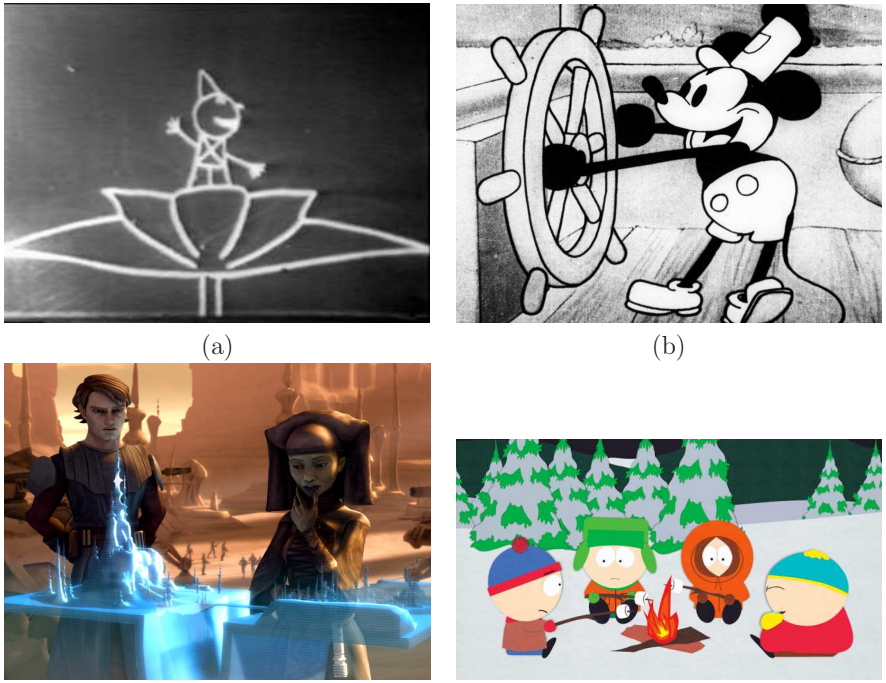
Toward convenience and portability Further motivations for accepting lower levels of realism are cost and portability. As shown in Figure 1.27, families were willing to gather in front of a television to watch free broadcasts in their homes, even though they could go to theaters and watch high-resolution, color, panoramic, and 3D movies at the time. Such tiny, blurry, black-and-white television sets seem comically intolerable with respect to our current expectations. The next level of portability is to carry the system around with you. Thus, the progression is from: 1) having to go somewhere to watch it, to 2) being able to watch it in your home, to 3) being able to carry it anywhere. Whether pictures, movies, phones, computers, or video games, the same progression continues. We can therefore expect the same for VR systems. At the same time, note that the gap is closing between these levels: The quality we expect from a portable device is closer than ever before to the version that requires going somewhere to experience it.



(a) (b) (c) (d)

Figure 1.25: A progression of special effects: (a) Arrival of a Train at La Ciotat Station, 1896. (b) A Trip to the Moon, 1902. (c) The movie 2001, from 1968. (d) *Gravity*, 2013.

Video games Motion pictures yield a passive, third-person experience, in contrast to video games which are closer to a first-person experience by allowing us to interact with him. Recall from Section 1.1 the differences between open-loop and closed-loop VR. Video games are an important step toward closed-loop VR, whereas motion pictures are open-loop. As shown in Figure 1.28, we see the same trend from simplicity to improved realism and then back to simplicity. The earliest games, such as Pong and Donkey Kong, left much to the imagination. *First-person shooter (FPS)* games such as Doom gave the player a first-person perspective and launched a major campaign over the following decade toward higher quality graphics and realism. Assassin's Creed shows a typical scene from a modern, realistic video game. At the same time, wildly popular games have emerged by focusing on simplicity. Angry Birds looks reminiscent of games from the 1980s, and Minecraft allows users to create and inhabit worlds composed of course blocks. Note that reduced realism often leads to simpler engineering requirements; in 2015, an advanced FPS game might require a powerful PC and graphics card, whereas simpler games would run on a basic smartphone. Repeated lesson: Don't assume that more realistic is better!



(a) (b) (c) (d)

Figure 1.26: A progression of cartoons: (a) Emile Cohl, Fantasmagorie, 1908. (b) Mickey Mouse in Steamboat Willie, 1928. (c) The Clone Wars Series, 2003. (d) South Park, 1997.

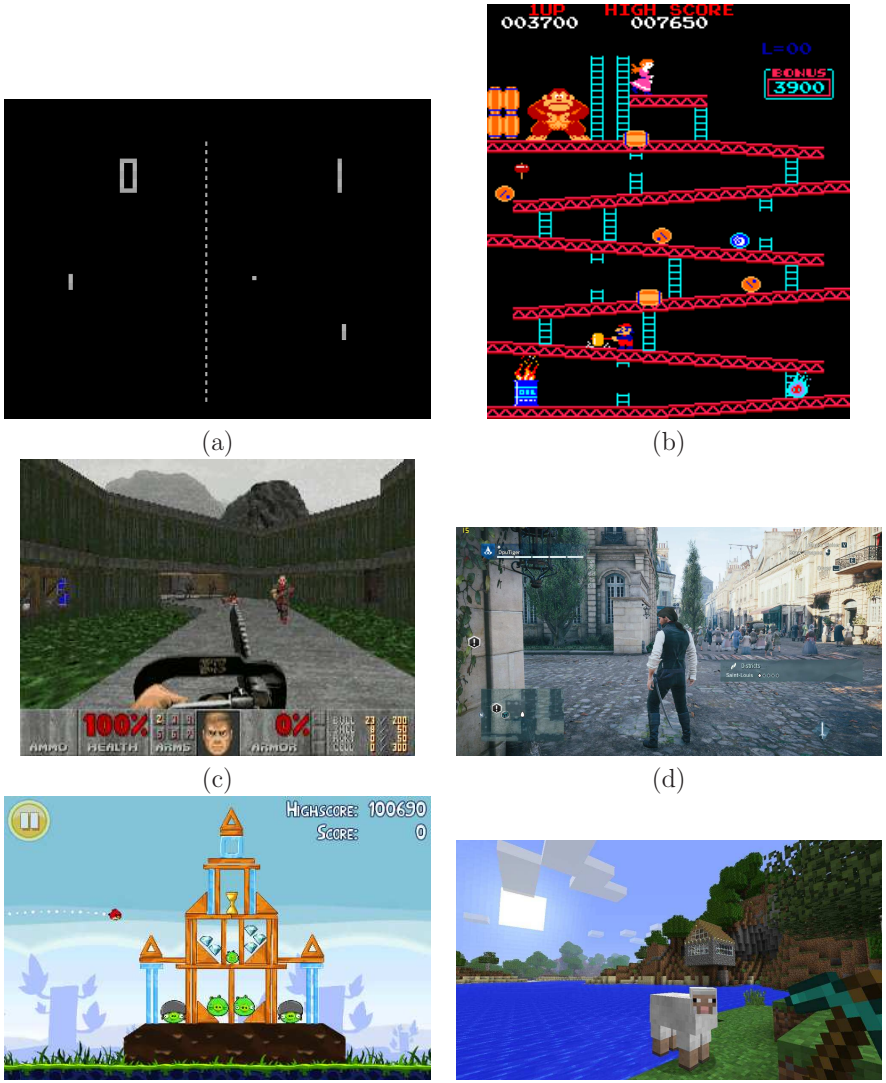
Beyond staring at a rectangle The concepts so far are still closely centered on staring at a rectangle that is fixed on a wall. Two important steps come next: 1) Presenting a separate picture to each eye to induce a "3D" effect. 2) Increasing the field of view so that the user is not distracted by the stimulus boundary. One way our brains infer the distance of objects from our eyes is by *stereopsis*. Information is gained by observing and matching features in the world that are visible to both the left and right eyes. The differences between their images on the retina yield cues about distances; keep in mind that there are many more such cues, which are explained in Section 6.1. The first experiment that showed the 3D effect of stereopsis was performed in 1838 by Charles Wheatstone in a system called the *stereoscope* (Figure 1.29(a)). By the 1930s, a portable version became a successful commercial product known to this day as the View-Master (Figure 1.29(b)). Pursuing this idea further led to Sensorama, which added motion pictures, sound, vibration, and even smells to the experience (Figure 1.29(c)). An unfortunate limitation of these designs is requiring that the viewpoint is fixed with

Figure 1.27: Although movie theaters with large screens were available, families were also content to gather around television sets that produced a viewing quality that would be unbearable by current standards, as shown in this photo from 1958.

respect to the picture. If the device is too large, then the user's head also becomes fixed in the world. An alternative has been available in movie theaters since the 1950s. Stereopsis was achieved when participants wore special glasses that select a different image for each eye using polarized light filters. This popularized *3D movies*, which are viewed the same way in the theaters today.

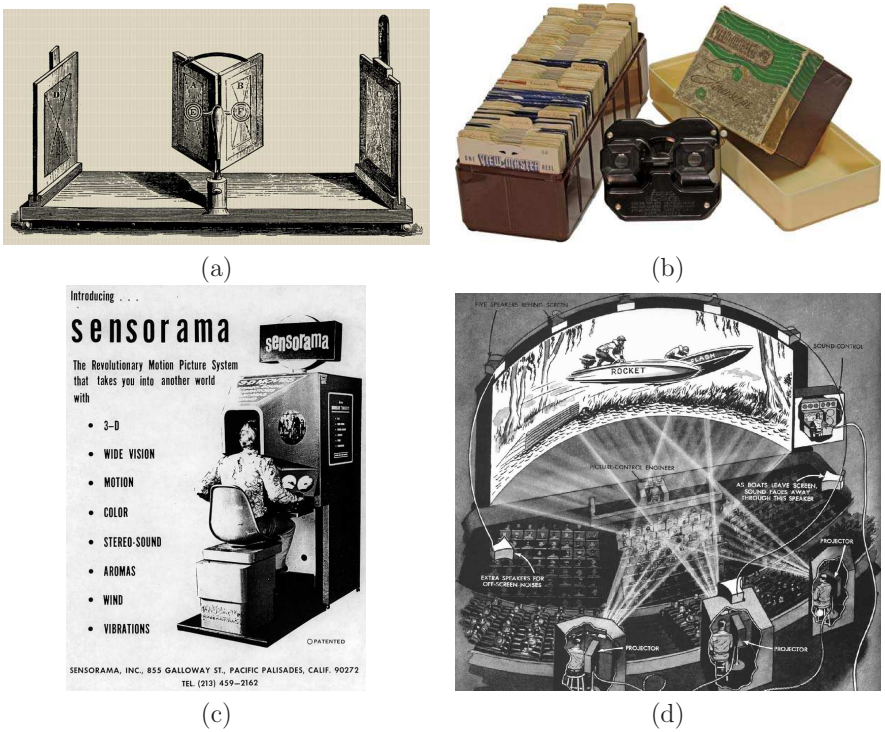
Another way to increase the sense of immersion and depth is to increase the field of view. The Cinerama system from the 1950s offered a curved, wide field of view that is similar to the curved, large LED (Light-Emitting Diode) displays offered today (Figure 1.29(d)). Along these lines, we could place screens all around us. This idea led to one important family of VR systems called the *CAVE*, which was introduced in 1992 at the University of Illinois [6] (Figure 1.30(a)). The user enters a room in which video is projected onto several walls. The CAVE system also offers stereoscopic viewing by presenting different images to each eye using polarized light and special glasses. Often, head tracking is additionally performed to allow viewpoint-dependent video to appear on the walls.

VR headsets Once again, the trend toward portability appears. An important step for VR was taken in 1968 with the introduction of Ivan Sutherland's



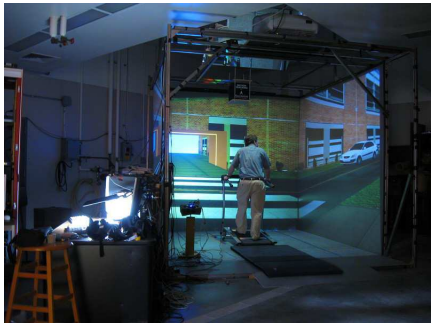
(a) (b) (c) (d) (e) (f)

Figure 1.28: A progression of video games: (a) Atari's Pong, 1972. (b) Nintendo's Donkey Kong, 1981. (c) id Software's Doom, 1993. (d) Ubisoft's Assassin's Creed Unity, 2014. (e) Rovio Entertainment's Angry Birds, 2009. (f) Markus "Notch" Persson's Minecraft, 2011.



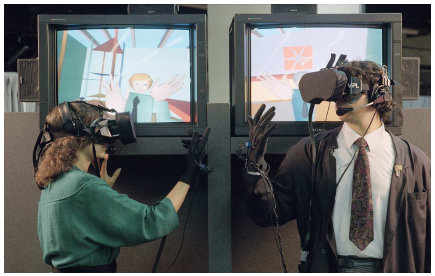
(a) (b) (c) (d)

Figure 1.29: (a) The first stereoscope, developed by Charles Wheatstone in 1838, used mirrors to present a different image to each eye; the mirrors were replaced by lenses soon afterward. (b) The View-Master is a mass-produced stereoscope that has been available since the 1930s. (c) In 1957, Morton Heilig's Sensorama added motion pictures, sound, vibration, and even smells to the experience. (d) In competition to stereoscopic viewing, Cinerama offered a larger field of view. Larger movie screens caused the popularity of 3D movies to wane in the 1950s.



(a)

(b)



(c)

(d)

(e)

(f)

Figure 1.30: (a) CAVE virtual environment, Illinois Simulator Laboratory, Beckman Institute, University of Illinois at Urbana-Champaign, 1992 (photo by Hank Kaczmarski). (b) Sutherland's Ultimate Display, 1968. (c) VPL Eyephones, 1980s. (d) Virtuality gaming, 1990s. (e) Nintendo Virtual Boy, 1995. (f) Oculus Rift, 2016.

Figure 1.31: Second Life was introduced in 2003 as a way for people to socialize through avatars and essentially build a virtual world to live in. Shown here is the author giving a keynote address at the 2014 Opensimulator Community Conference. The developers build open source software tools for constructing and hosting such communities of avatars with real people behind them.

Ultimate Display, which leveraged the power of modern displays and computers (Figure 1.30(b)) [22, 23]. He constructed what is widely considered to be the first VR headset. As the user turns his head, the images presented on the screen are adjusted to compensate so that the virtual objects appear to be fixed in space. This yielded the first glimpse of an important concept in this book: The *perception of stationarity*. To make an object appear to be stationary while you move your sense organ, the device producing the stimulus must change its output to compensate for the motion. This requires sensors and tracking systems to become part of the VR system. Commercial VR headsets started appearing in the 1980s with Jaron Lanier's company VPL, thereby popularizing the image of *goggles and gloves*; Figure 1.30(c). In the 1990s, VR-based video games appeared in arcades (Figure 1.30(d)) and in home units (Figure 1.30(e). The experiences were not compelling or comfortable enough to attract mass interest. However, the current generation of VR headset leverages the widespread availability of high resolution screens and sensors, due to the smartphone industry, to offer lightweight, low-cost, high-field-of-view headsets, such as the Oculus Rift (Figure 1.30(f)). This has greatly improved the quality of VR experiences while significantly lowering the barrier of entry for developers and hobbyists. This also caused a recent flood of interest in VR technology and applications.

Bringing people together We have so far neglected an important aspect, which is human-to-human or social interaction. We use formats such as a live theater performance, a classroom, or a lecture hall for a few people to communicate with or entertain a large audience. We write and read novels to tell stories to each other. Prior to writing, skilled storytellers would propagate experiences to others, including future generations. We have communicated for centuries by writing letters to each other. More recent technologies have allowed us to interact directly without delay. The audio part has been transmitted through telephones for over a century, and now the video part is transmitted as well through videoconferencing over the Internet. At the same time, simple text messaging has become a valuable part of our interaction, providing yet another example of a preference for decreased realism. Communities of online users who interact through text messages over the Internet have been growing since the 1970s. In the context of games, early Multi-User Dungeons (MUDs) grew into Massively Multiplayer Online Games (MMORPGs) that we have today. In the context of education, the PLATO system from the University of Illinois was the first computer-assisted instruction system, which included message boards, instant messaging, screen sharing, chat rooms, and emoticons. This was a precursor to many community-based, on-line learning systems, such as the Khan Academy and Coursera. The largest amount of online social interaction today occurs through Facebook apps, which involve direct communication through text along with the sharing of pictures, videos, and links.

Returning to VR, we can create *avatar* representations of ourselves and "live" together in virtual environments, as is the case with Second Life and Opensimulator 1.31. Without being limited to staring at rectangles, what kinds of societies will emerge with VR? Popular science fiction novels have painted a thrilling, yet dystopian future of a world where everyone prefers to interact through VR [5, 7, 21]. It remains to be seen what the future will bring.

As the technologies evolve over the years, keep in mind the power of simplicity when making a VR experience. In some cases, maximum realism may be important; however, leaving much to the imagination of the users is also valuable. Although the technology changes, one important invariant is that humans are still designed the same way. Understanding how our senses, brains, and bodies work is crucial to understanding the fundamentals of VR systems.

Further reading

Each chapter of this book concludes with pointers to additional, related literature that might not have been mentioned in the preceding text. Numerous books have been written on VR. A couple of key textbooks that precede the consumer VR revolution are *Understanding Virtual Reality* by W. R. Sherman and A. B. Craig, 2002 [19] and *3D User Interfaces* by D. A. Bowman et al., 2005 [2]. Books based on the current technology include [11, 12]. For a survey of the concept of reality, see [26]. For recent coverage of *augmented reality* that is beyond the scope of this book, see [18].

A vast amount of research literature has been written on VR. Unfortunately, there is a considerable recognition gap in the sense that current industry approaches to consumer VR appear to have forgotten the longer history of VR research. Many of the issues being raised today and methods being introduced in industry were well addressed decades earlier, albeit with older technological components. Much of the earlier work remains relevant today and is therefore worth studying carefully. An excellent starting place is the *Handbook on Virtual Environments*, 2015 [8], which contains dozens of recent survey articles and thousands of references to research articles. More recent works can be found in venues that publish papers related to VR. Browsing through recent publications in these venues may be useful: IEEE Virtual Reality (IEEE VR), IEEE International Conference on Mixed and Augmented Reality (ISMAR), ACM SIGGRAPH Conference, ACM Symposium on Applied Perception, ACM SIGCHI Conference, IEEE Symposium of 3D User Interfaces, *Journal of Vision*, *Presence: Teleoperators and Virtual Environments*.

Bibliography

[1] Z. M. Aghajan, L. Acharya, J. J. Moore, J. D. Cushman, C. Vuong, and M. R. Mehta. Impaired spatial selectivity and intact phase precession in two-dimensional virtual reality. *Nature Neuroscience*, 18(1):121–128, 2015.

[2] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. *3D User Interfaces*. Addison-Wesley, Boston, MA, 2005.

[3] N. C. Burbules. Rethinking the virtual. In J. Weiss, J. Nolan, and P. Trifonas, editors, *The International Handbook of Virtual Learning Environments*, pages 3–24. Kluwer Publishers, Dordrecht, 2005.

[4] G. Chen, J. A. King, N. Burgess, and J. O'Keefe. How vision and movement combine in the hippocampal place code. *Proceedings of the National Academy of Science USA*, 110(1):378–383, 2013.

[5] E. Cline. *Ready Player One*. Random House, 2011.

[6] C. Cruz-Neira, D. J. SAndin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The CAVE: Audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72, 1992.

[7] W. Gibson. *Neuromancer*. Ace Books, 1984.

[8] K. S. Hale and K. M. Stanney. *Handbook of Virtual Environments, 2nd Edition*. CRC Press, Boca Raton, FL, 2015.

[9] C. D. Harvey, F. Collman, D. A. Dombeck, and D. W. Tank. Intracellular dynamics of hippocampal place cells during virtual navigation. *Nature*, 461:941–946, 2009.

[10] P. Hoberman, D. M. Krum, E. A. Suma, and M. Bolas. Immersive training games for smartphone-based head mounted displays. In *IEEE Virtual Reality Short Papers and Posters*, 2012.

[11] J. Jerald. *The VR Book*. Association of Computer Machinery and Morgan & Claypool Publishers, 2015.

[12] J. Linowes. *Unity Virtual Reality Projects*. Packt, Birmingham, UK, 2015.

[13] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. In *Proceedings of Telemanipulator and Telepresence Technologies*, 1994.

[14] E. I. Moser, E. Kropff, and M.-B. Moser. Place cells, grid cells, and the brain's spatial representation system. *Annual Reviews of Neuroscience*, 31:69–89, 2008.

[15] D. Nitz. A place for motion in mapping. *Nature Neuroscience*, 18:6–7, 2010.

[16] J. O'Keefe and J. Dosytovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34(1):171–175, 1971.

[17] J. L. Olson, D. M. Krum, E. A. Suma, and M. Bolas. A design for a smartphone-based head mounted display. In *Proceedings IEEE Virtual Reality Conference*, pages 233–234, 2011.

[18] D. Schmalsteig and T. Höllerer. *Augmented Reality: Principles and Practice*. Mendeley Ltd., London, 2015.

[19] W. R. Sherman and A. B. Craig. *Understanding Virtual Reality: Interface, Application, and Design*. Morgan Kaufmann, San Francisco, CA, 2002.

[20] A. Steed and S. Julier. Design and implementation of an immersive virtual reality system based on a smartphone platform. In *Proceedings IEEE Symposium on 3D User Interfaces*, 2013.

[21] N. Stephenson. *Snow Crash*. Bantam Books, 1996.

[22] I. E. Sutherland. The ultimate display. In *Proceedings of the IFIP Congress*, pages 506–508, 1965.

[23] I. E. Sutherland. A head-mounted three dimensional display. In *Proceedings of AFIPS*, pages 757–764, 1968.

[24] K. Thurley and A. Ayaz. Virtual reality systems for rodents. *Current Zoology*, 63(1), 2017.

[25] S. T. von Soemmerring. *Über das Organ der Seele*. Königsberg, 1796. With afterword by Immanuel Kant.

[26] J. Westerhoff. *Reality: A Very Short Introduction*. Oxford University Press, Oxford, UK, 2011.