Computer Graphics and Animation

Lecturer: Bijay Mishra (बिजय मिश्र)

 **Contact No: 9841695609**

 **Email ID: biizay@gmail.com**

 **@jijibisha**

Putting your Mobile away and paying attention to those talking to you? There's **App** for that, it's called **RESPECT**!

Computer Graphics and Animation

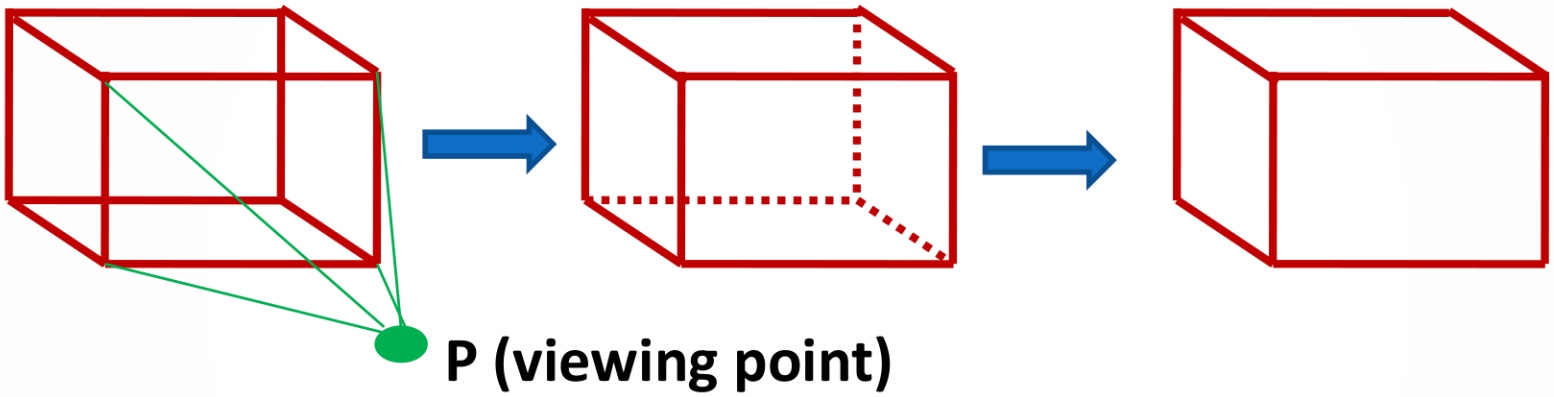
Unit 4: Visible Surface Determination and Computer Graphics Algorithm

BIJAY MISHRA

Visible Surface Determination

Visible Surface Detection

Given a *scene* and a *projection*, What can we see?



- Given a set of 3D object and viewing point
- This process find which lines or parts of the object are visible from the viewing point, so that it can draw only those part.

Visible Surface Detection

- It is the process of identifying those parts of a scene that are visible from a chosen viewing position.
- There are numerous algorithms for efficient identification of visible objects for different types of applications.
- These various algorithms are referred to as ***visible-surface detection methods/algorithms.***
- Visible surface detection methods are classified according to whether they deal with objects or with their projected images.
- Visible surface detection algorithms are broadly classified as:
  1. Object-Space Methods (OSM)
  2. Image-Space Methods (ISM)

Visible Surface Detection

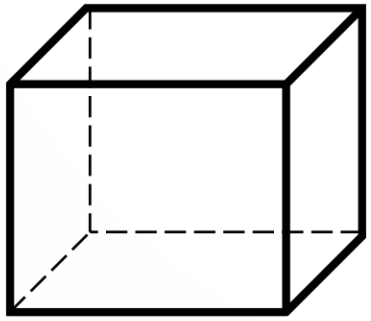
- Visible surface detection algorithms are broadly classified as:

1. **Object-Space Methods (OSM)**

   Compares objects and parts of objects to each other within the scene definition to determine which surfaces are visible

2. **Image-Space Methods (ISM)**

   Visibility is decided point-by-point at each pixel position on the projection plane



Object space

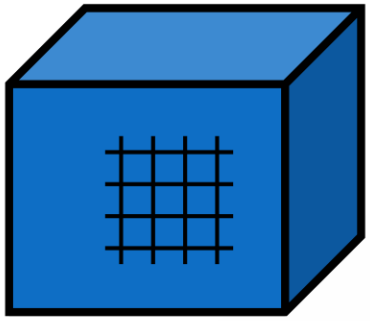


Image space

Visible Surface Detection

Object Space Methods:

- Deal with object definition
- *Determine which part of the object are visible*
- Compares objects and parts of objects to each other within the scene definition to determine which surface as a whole we should label as visible.
- Example: *Back-face removal method*

Visible Surface Detection

Object Space Methods

- Object space algorithms do their work on the objects themselves before they are converted to pixels in the frame buffer.
- The resolution of the display device is irrelevant here as this calculation is done at the mathematical level of the objects

- Pseudo code…
  - for each object a in the scene
    - determine which parts of object a are visible
    - draw these parts in the appropriate color

- Involves comparing the polygons in object a to other polygons in a and to polygons in every other object in the scene.

Visible Surface Detection

Image Space Methods:

- Deal with projected image
- *Determine per pixel which point of an object is visible*
- Visibility is decided point by point at each pixel position on the projection plane.
- Image space methods are by far the more common.
- Examples: *Depth-buffer method*, *Scan-line method*, *Area-subdivision method*

Visible Surface Detection

Image Space Methods

- Image space algorithms do their work as the objects are being converted to pixels in the frame buffer.
- The resolution of the display device is important here as this is done on a pixel by pixel basis.

- Pseudo code…
  - for each pixel in the frame buffer
    - determine which polygon is closest to the viewer at that pixel location
    - color the pixel with the color of that polygon at that location

Visible Surface Detection

Object Space Methods	Image Space Methods
1. It is object based method. It concentrates on geometrical relation among objects in the scene.	1. It is a pixel-based method. It is concerned with the final image, what is visible within each raster pixel.
2. Here surface visibility is determined.	2. Here line visibility or point visibility is determined.
3. It is performed at the precision with which each object is defined, No resolution is considered.	3. It is performed using the resolution of the display device.
4. Calculations are not based on the resolution of the display so change of object can be easily adjusted.	4. Calculations are resolution base, so the change is difficult to adjust.
5. These were developed for vector graphics system.	5. These are developed for raster devices.

Visible Surface Detection

Object Space Methods	Image Space Methods
6. Object-based algorithms operate on continuous object data.	6. These operate on object data.
7. Vector display used for object method has large address space.	7. Raster systems used for image space methods have limited address space.
8. Object precision is used for application where speed is required.	8. There are suitable for application where accuracy is required.
9. It requires a lot of calculations if the image is to enlarge.	9. Image can be enlarged without losing accuracy.
10. If the number of objects in the scene increases, computation time also increases.	10. In this method complexity increase with the complexity of visible parts.

Visible Surface Detection

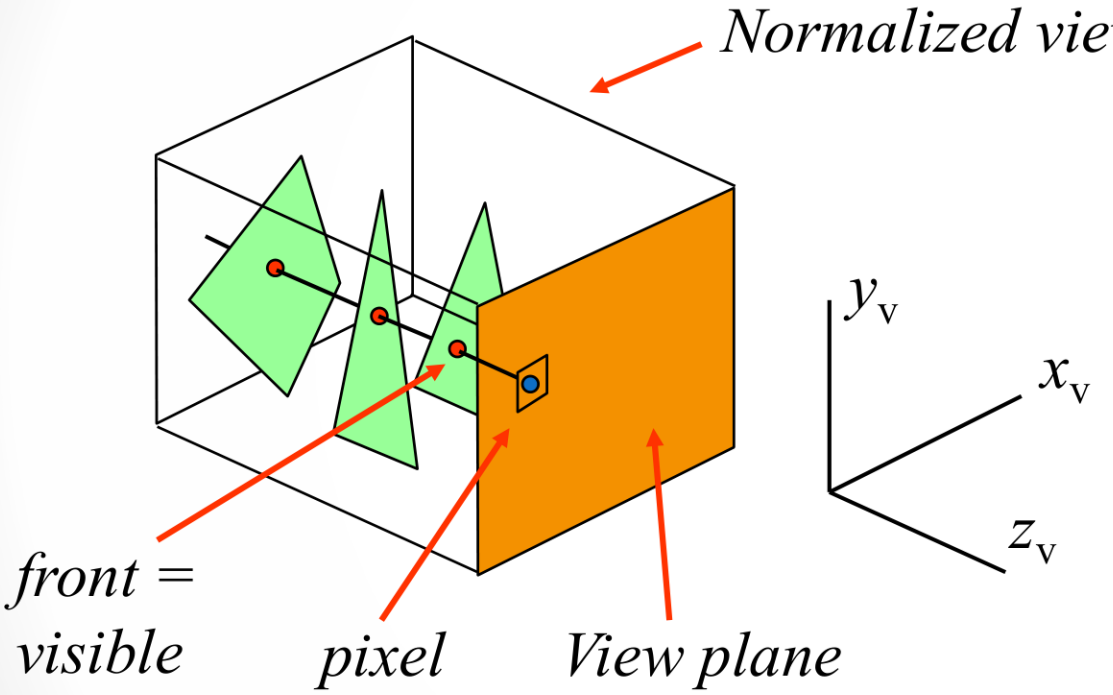
List Priority Algorithms

- This is a *hybrid model* that combines both object and image precision operations.
- Here, depth comparison & object splitting are done with object precision and scan conversion (which relies on ability of graphics device to overwrite pixels of previously drawn objects) is done with image precision.
- Examples: *Depth-Sorting method*, *BSP-tree method*

Depth – Buffer (Z – Buffer Method)

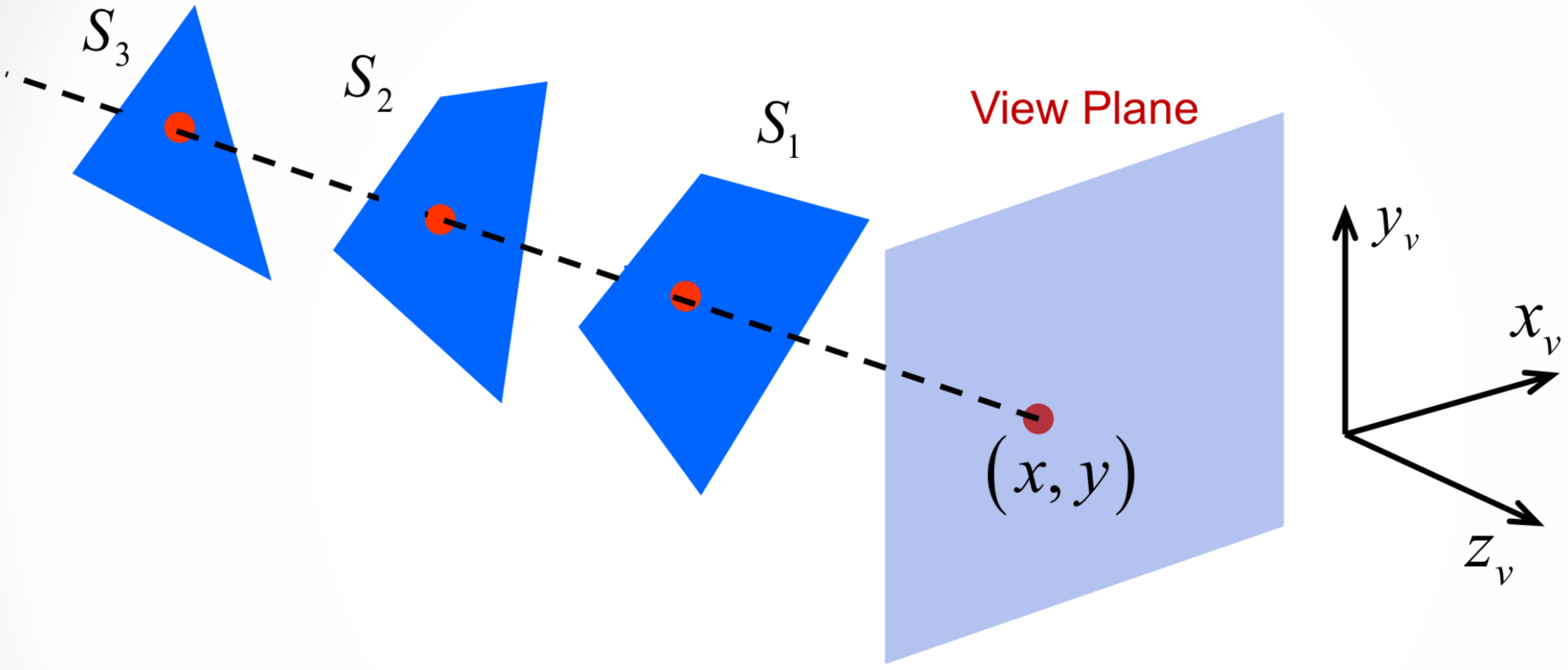
- A commonly used ***image-space*** approach to detecting visible surfaces is the ***depth-buffer method***, which compares surface depths at each pixel position on the projection plane.
- Also called ***z-buffer method*** since depth usually measured along z-axis.
- Compares **surface depth values** throughout a scene **for each pixel position** on the **projection plane**
- **Each surface of a scene is processed separately**
- The algorithm is usually applied to **scenes containing only polygon surfaces**
- Each (x, y, z) position on a polygon surface corresponds to the projection point (x, y) on the view plane.

Depth – Buffer (Z – Buffer Method)



Algorithm:
Draw polygons, Remember the color most in front.

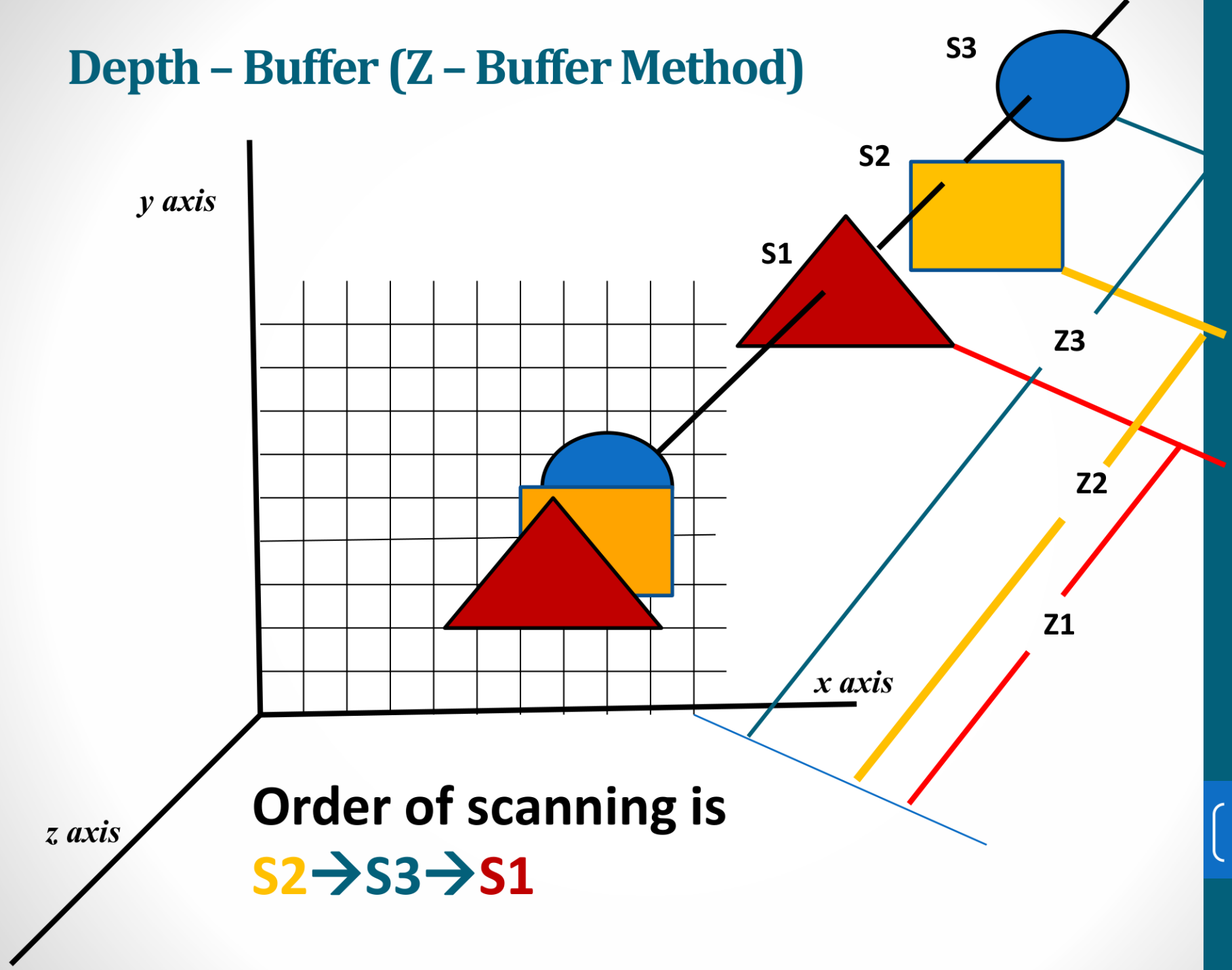
Depth – Buffer (Z – Buffer Method)



Three surfaces overlapping pixel position *(x, y)* on the view plane.

The visible surface, S_1, has the smallest depth value.

Depth – Buffer (Z – Buffer Method)



Order of scanning is

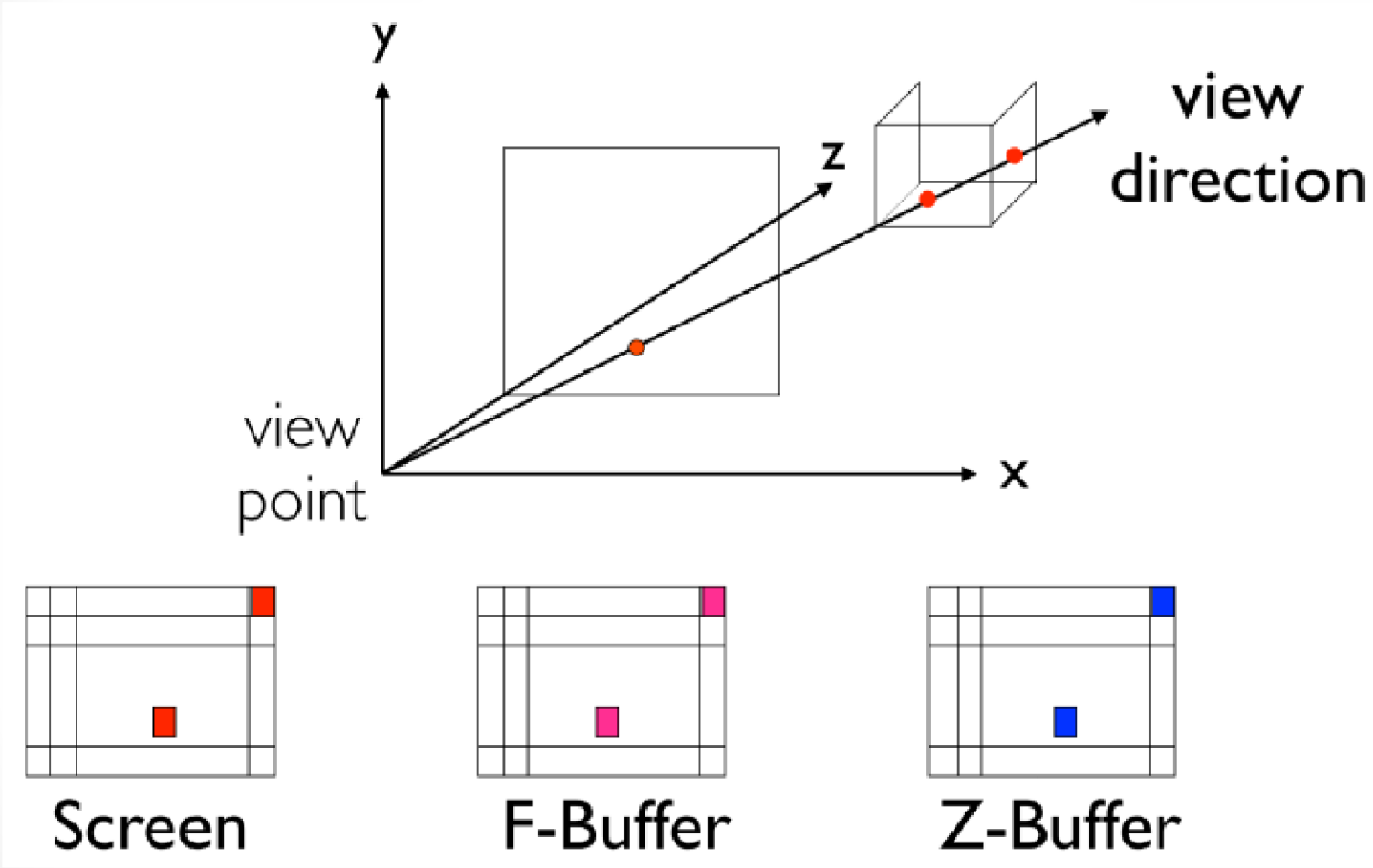
S2→S3→S1

Depth – Buffer (Z – Buffer Method)

This method requires two buffers:

- **Depth Buffer**
  - Also called as ***Z-buffer***
  - Store **depth (z) values** for each (x, y) pixel position
  - All positions are initialized to minimum depth
    - Usually 0 – most distant depth from the view plane
- **Refresh Buffer**
  - Also called as ***frame buffer***/***image buffer***
  - Stores the **surface-intensity values** or **color values** for each pixel position
  - All positions are initialized to the background intensity

Depth – Buffer (Z – Buffer Method)



Depth – Buffer (Z – Buffer Method)

Z - Buffer Algorithm **(Pseudocode)**

Initialize each pixel (x, y) of depth buffer and frame buffer (color):

$depthBuf(x, y) = 1.0$; // Z is normalized to [0,1.0]

$frameBuf(x, y) =$ background color; // initialize to background

for each polygon { // traverse all polygons

  for each pixel $(x, y) \in$ polygon { // rasterization

    if $(z(x, y) > depthBuf(x, y))$ { // check closer pixel

      get (compute) $color(x, y)$; // compute color of pixel

      $depthBuf(x, y) = z(x, y)$; $frameBuf(x, y) = color(x, y)$;

}}}

Depth – Buffer (Z – Buffer Method)

Algorithm:

1. Initialize the depth buffer and frame buffer so that for all buffer positions (x, y),
   a. depthBuff (x, y) = 1.0, frameBuff (x, y) = backgndColor

2. Process each polygon in a scene, one at a time, as follows:
   a. For each projected (x, y) pixel position of a polygon, calculate the depth z (if not already known).
   b. If z < depthBuff (x, y), compute the surface color at that position and set depthBuff (x, y) = z, frameBuff (x, y) = surfColor (x, y)

After all surfaces have been processed, the **depth buffer contains depth values for the visible surfaces** and the **frame buffer contains the corresponding color values for those surfaces**

Depth – Buffer (Z – Buffer Method)

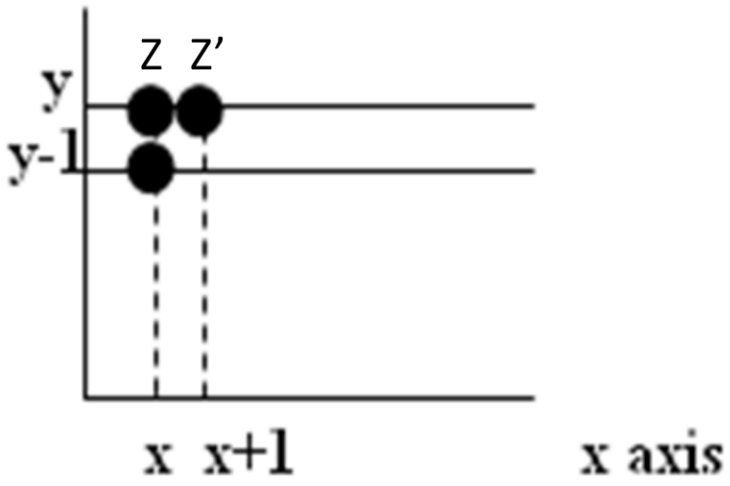
- After all surfaces have been processed the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces.

From plane equation, depth at position (x, y):
$z = (-Ax - By - D) / C$

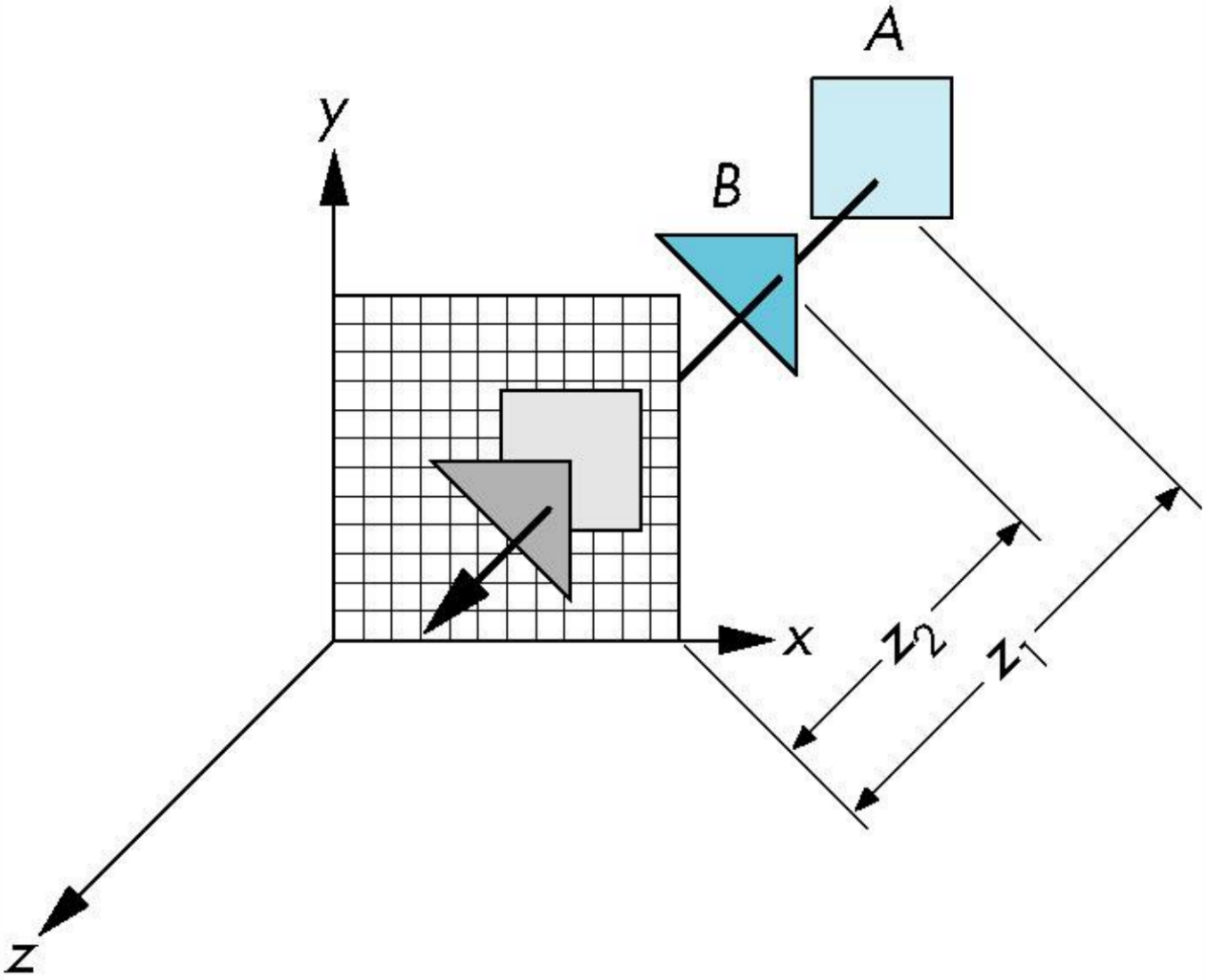
Incrementally across scan line (x+1, y):
$z' = (-A(x+1) - By - D) / C$
$= (-Ax - By - D) / C - A/C$
$= z - A/C$

Incrementally between scan lines (x', y+1):
$z' = (-A(x') - B(y+1) - D) / C$

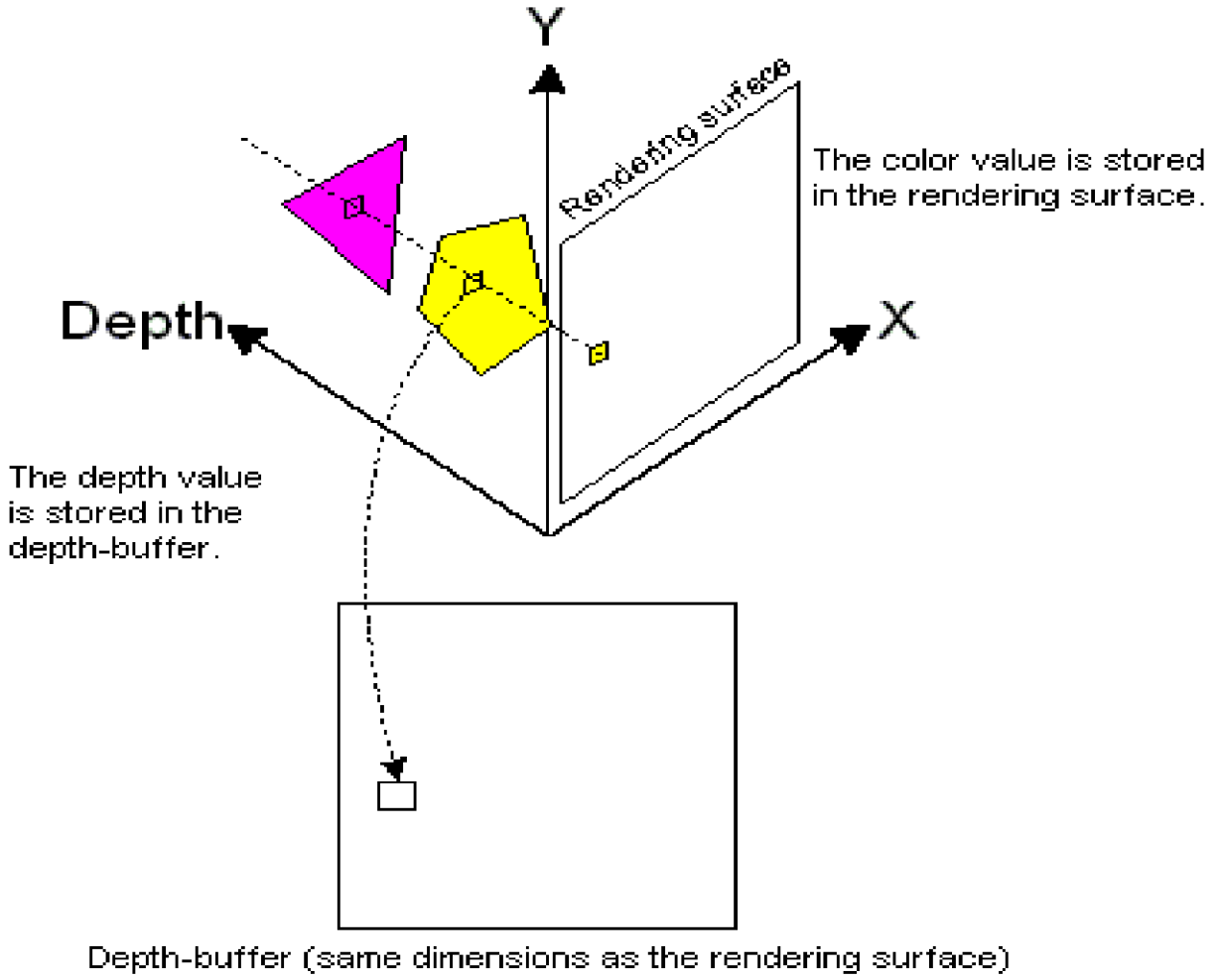
using $x' = x + 1/m$
$= (-A(x + 1/m) - B(y+1) - D) / C$
$= z - (A/m + B) / C$



Depth – Buffer (Z – Buffer Method)

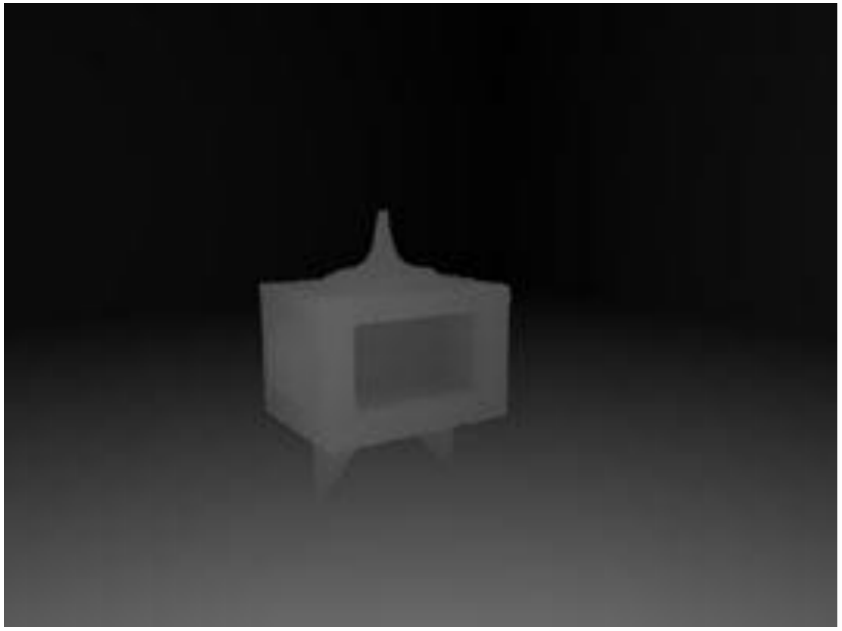


Depth – Buffer (Z – Buffer Method)



Depth – Buffer (Z – Buffer Method)

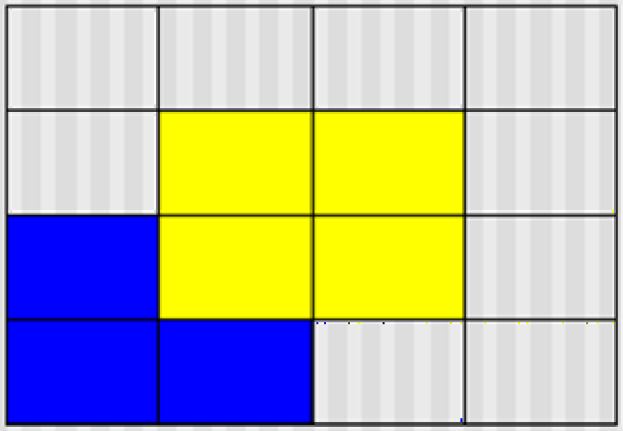
Color buffer



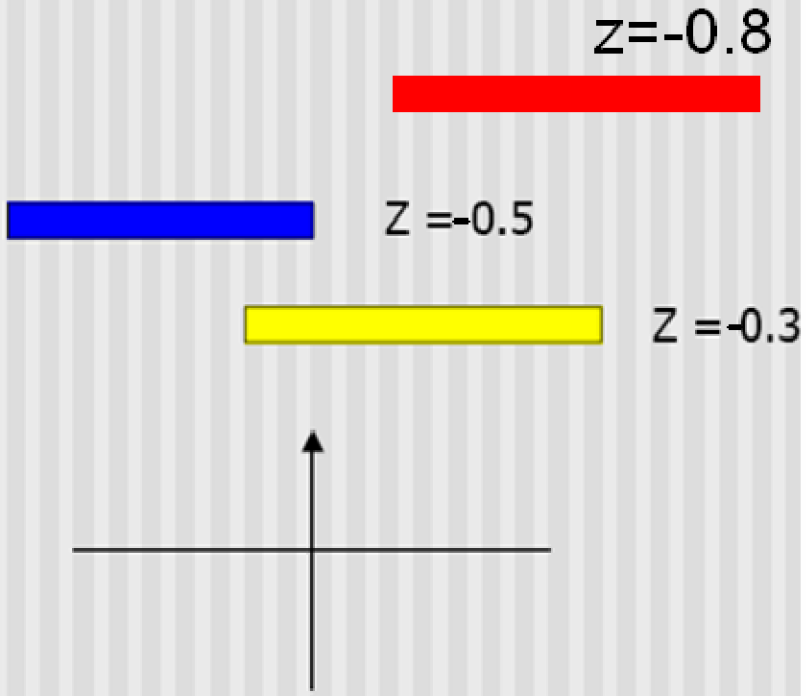
Depth buffer

DAM Entertainment

Depth – Buffer (Z – Buffer Method)



Correct Final image



Top View

Depth – Buffer (Z – Buffer Method)

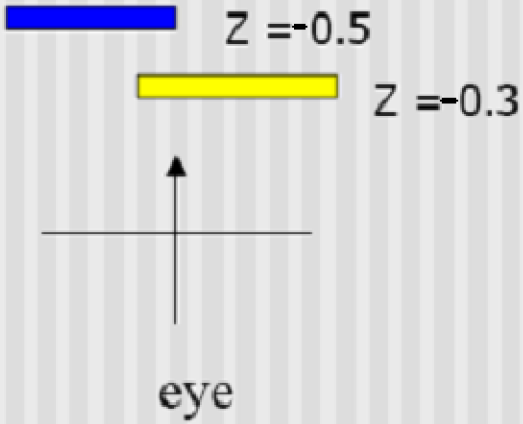
Step 1: Initialize the depth buffer

-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0

Depth – Buffer (Z – Buffer Method)

Step 2: Draw the blue polygon (assuming the program draws blue polyon first – the order does not affect the final result any way).

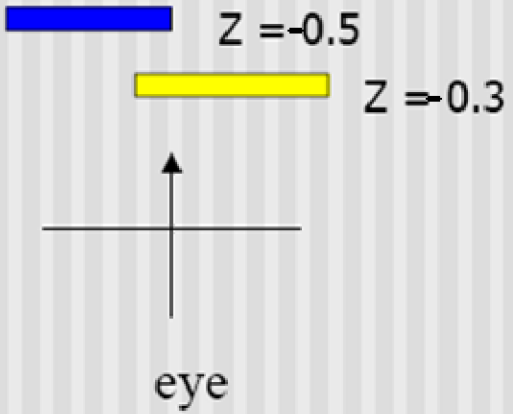
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0
-0.5	-0.5	-1.0	-1.0



Depth – Buffer (Z – Buffer Method)

Step 3: Draw the yellow polygon

-1.0	-1.0	-1.0	-1.0
-1.0	-0.3	-0.3	-1.0
-0.5	-0.3	-0.3	-1.0
-0.5	-0.5	-1.0	-1.0

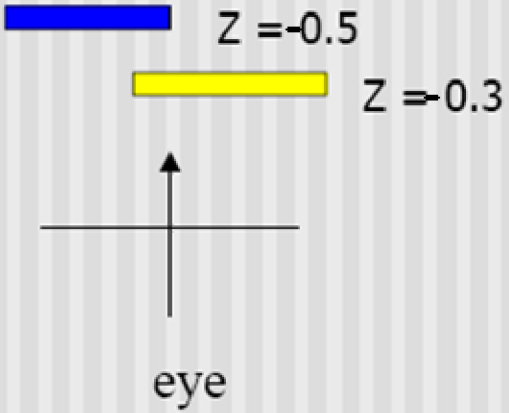


z-buffer drawback: wastes resources by rendering a face and then drawing over it

Depth – Buffer (Z – Buffer Method)

Step 4: Draw the red polygon

-1.0	-1.0	-0.8	-0.8
-1.0	-0.3	-0.3	-0.8
-0.5	-0.3	-0.3	-1.0
-0.5	-0.5	-1.0	-1.0



z-buffer drawback: wastes resources by rendering a face and then drawing over it

Depth – Buffer (Z – Buffer Method)

Advantages

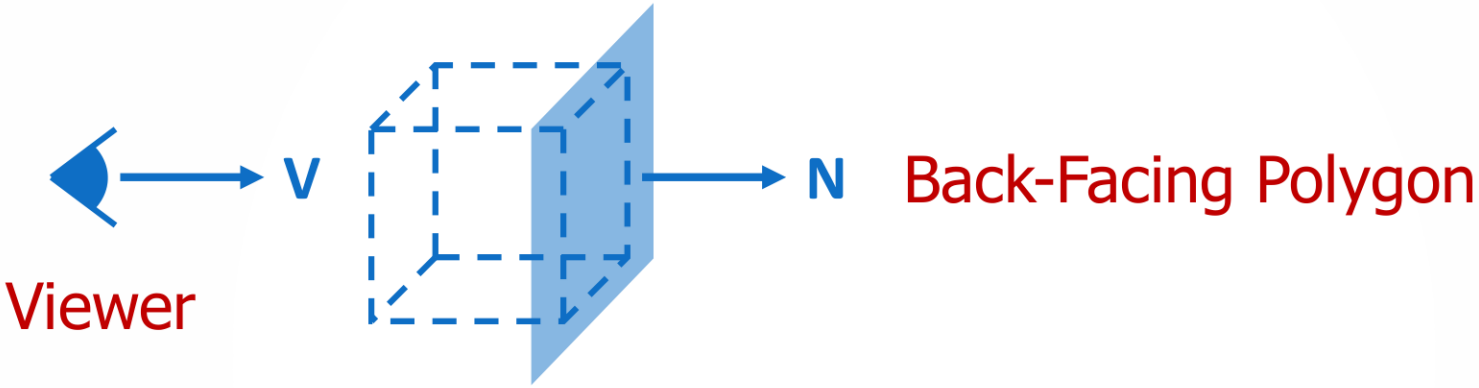
+ Easy to implement
+ Hardware supported
+ Polygons can be processed in arbitrary order
+ Requires no sorting of surface
+ Fast

Disadvantages

- Costs memory
- Color calculation sometimes done multiple times
- Transparency is tricky

Back – Face Detection Method

- Surfaces may be Back-Facing



Polygon is back-facing to the viewer if $V \cdot N > 0$

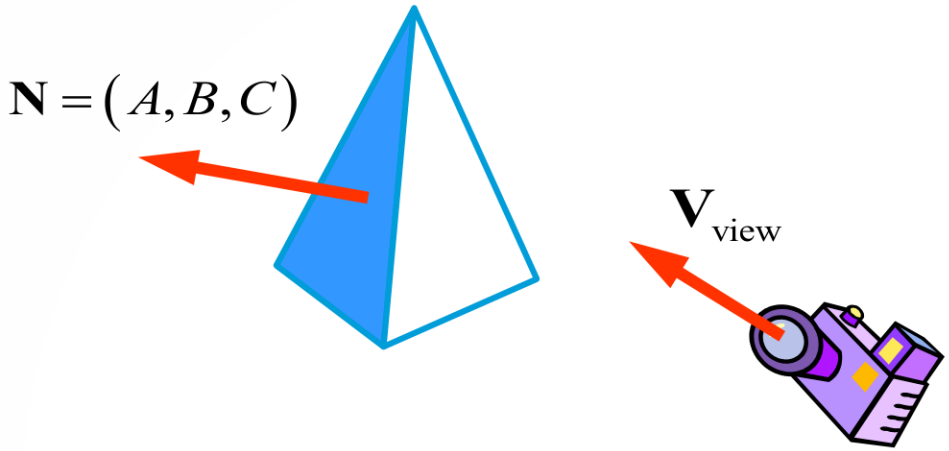
Back – Face Detection Method

- In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces which are opposite to the viewer (back faces).
- These back faces contribute to approximately half of the total number of surfaces.
- Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test.

Back – Face Detection Method

- Each surface has a normal vector. If this vector is pointing in the direction of the center of projection it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer.
- The test is very simple, if the z component of the normal vector is positive, then, it is a back face. If the z component of the vector is negative, it is a front face.

Back – Face Detection Method



- A point (x, y, z) is "inside" a surface with plane parameters A, B, C, and D if

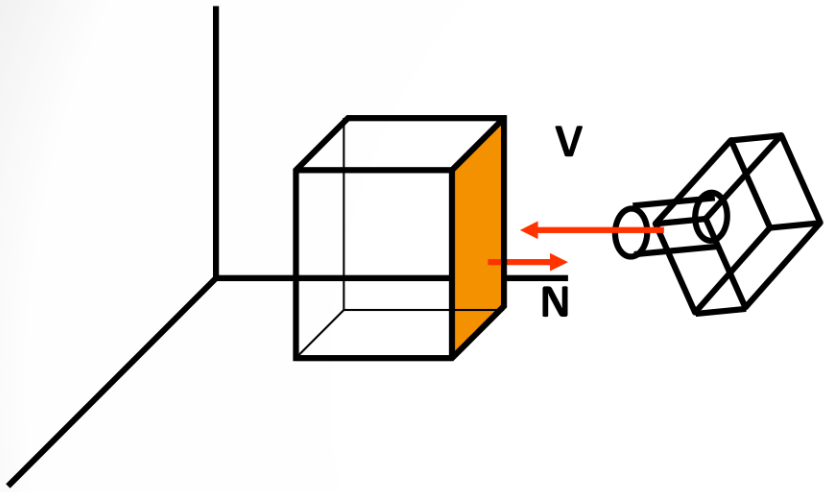
$$Ax + By + Cz + D < 0$$

- A point (x, y, z) is "outside" a surface with plane parameters A, B, C, and D if

$$Ax + By + Cz + D > 0$$

Back – Face Detection Method

The polygon is a front face if $\mathbf{V} \cdot \mathbf{N} < 0 : \text{front face}$



Where,

- V is a vector in the viewing direction from the eye (camera)
- N is the normal vector to a polygon surface

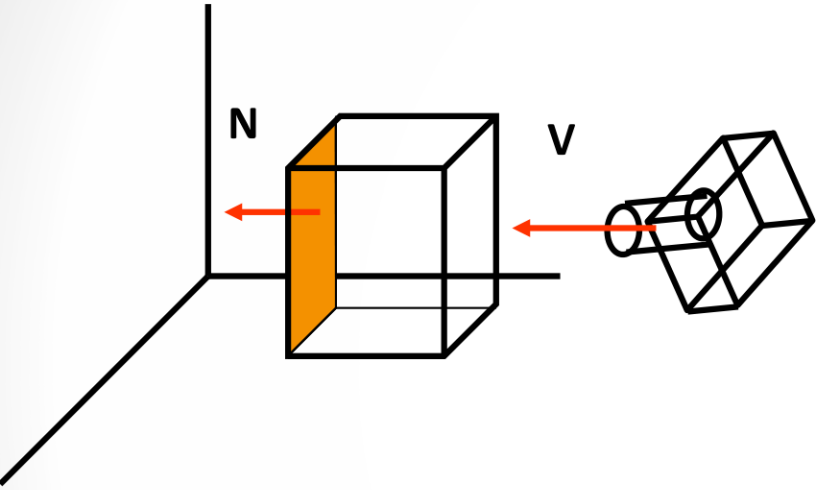
$$\vec{N} = A\hat{i} + B\hat{j} + C\hat{z}$$

$$\vec{V} = 0\hat{i} + 0\hat{j} + -1\hat{z}$$

$$\Rightarrow \vec{V} . \vec{N} = -C$$

Back – Face Detection Method

The polygon is a back face if $\mathbf{V} \cdot \mathbf{N} > 0 : \text{back face}$



Where,

- V is a vector in the viewing direction from the eye (camera)
- N is the normal vector to a polygon surface

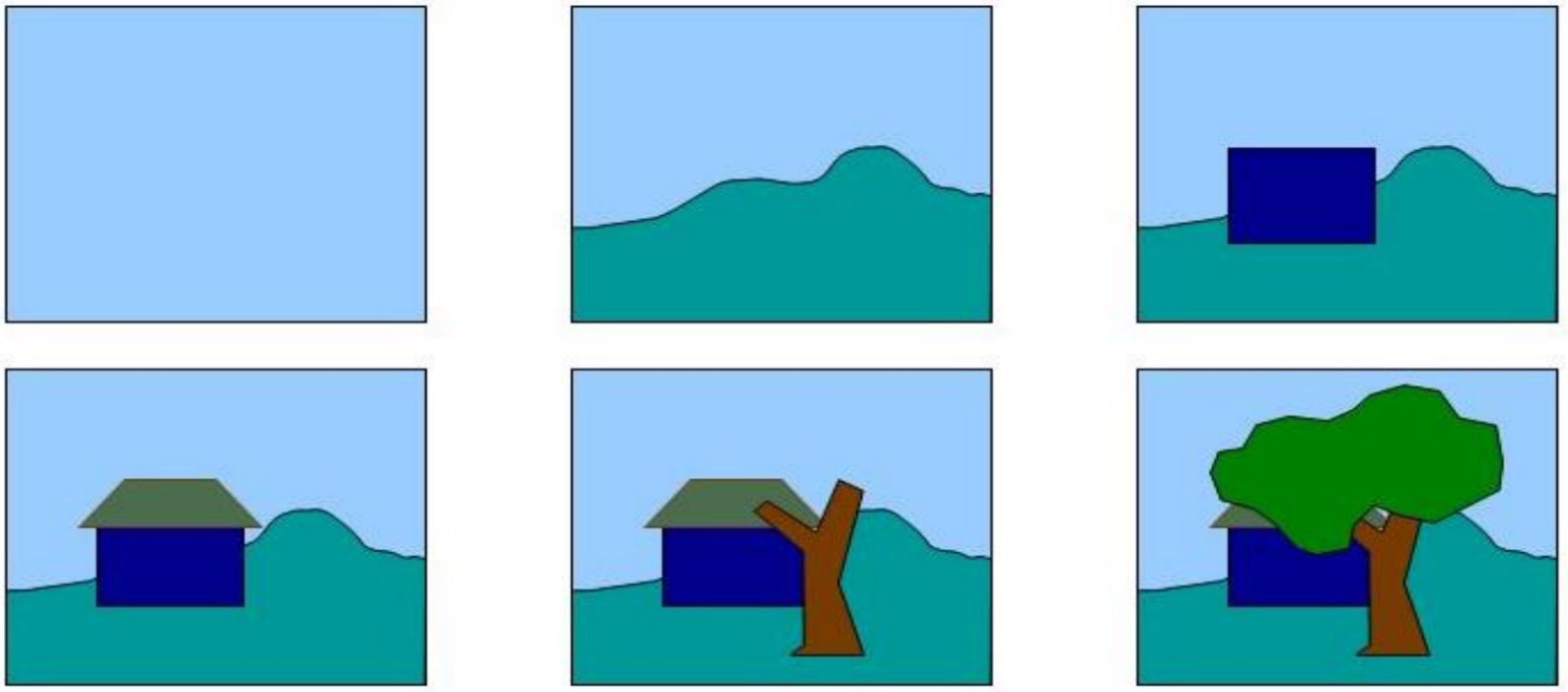
$$\vec{N} = -A\hat{i} + -B\hat{j} + -C\hat{z}$$

$$\vec{V} = 0\hat{i} + 0\hat{j} + -1\hat{z}$$

$$\Rightarrow \vec{V}.\vec{N} = C$$

The Painter's Algorithm (Depth Sort)

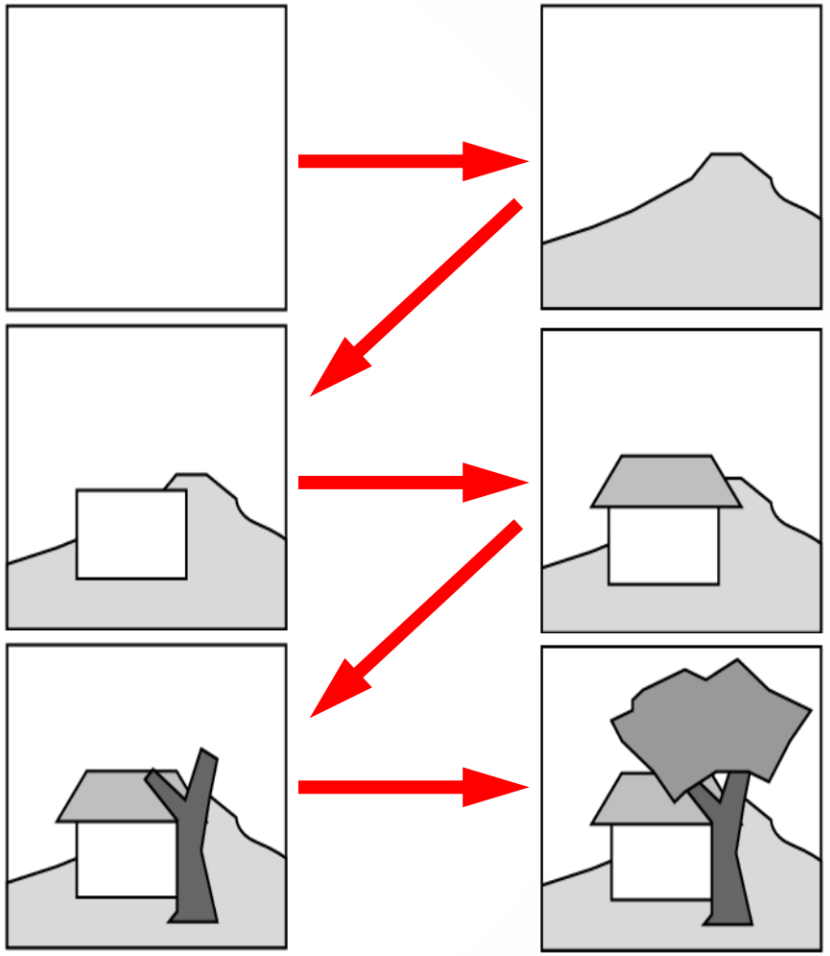
- This method uses both ***object space*** and ***image space*** method.
- In this method the surface representation of 3D object are sorted in of decreasing depth from viewer.
- Then, sorted surface are scan converted in order starting with surface of greatest depth for the viewer.



The Painter's Algorithm (Depth Sort)

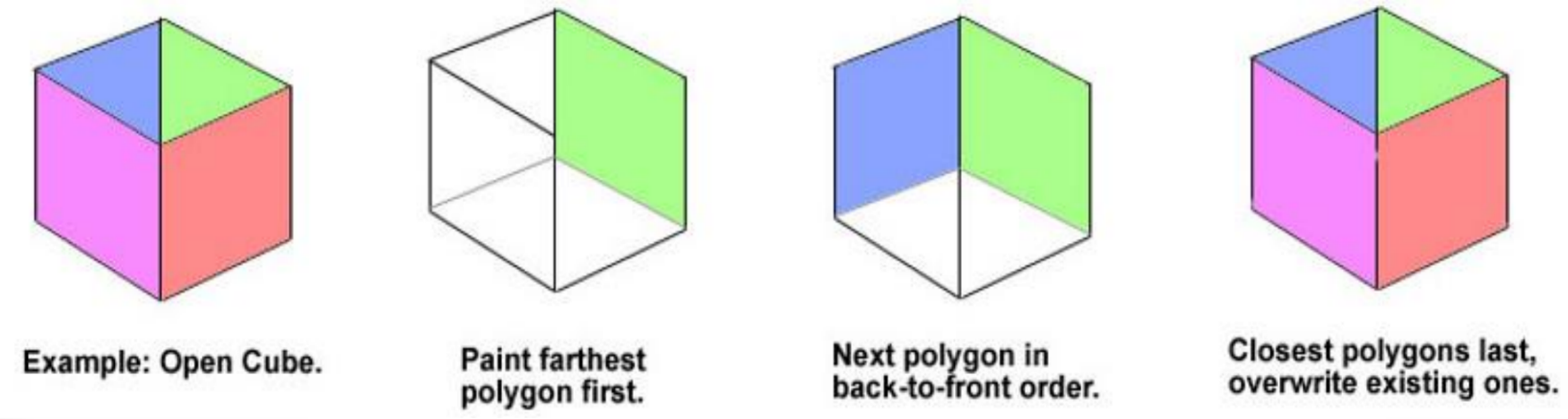
- **Main Idea**
  - A painter creates a picture by drawing background scene elements before foreground ones

- **Requirements**
  - Draw polygons in back-to-front order
  - Need to **sort** the polygons by depth order to get a correct image



from Shirley

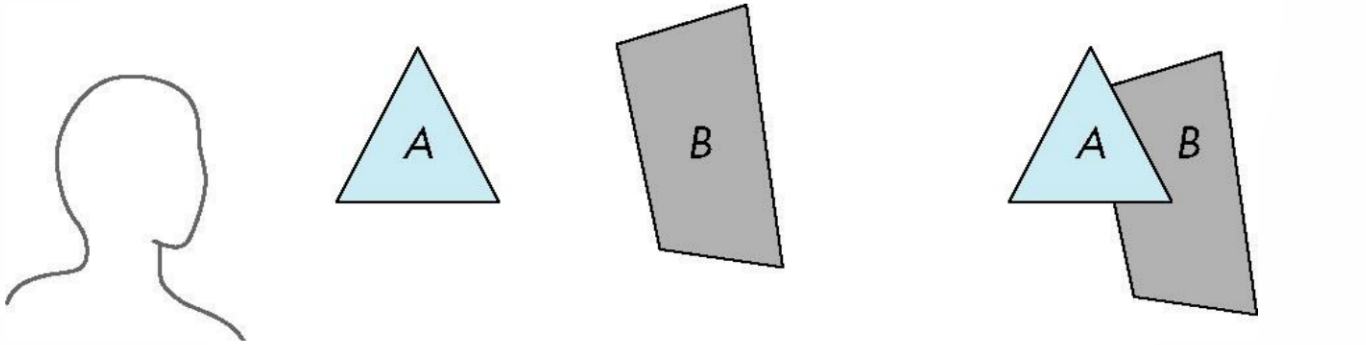
The Painter's Algorithm (Depth Sort)



- Sort polygons from back to front
- Draw each sorted polygon, painting over the pixels of existing ones.
- Surface are sorted in order of decreasing depth
- Surface are scan converted in order, starting with surface of greatest depth.

The Painter's Algorithm (Depth Sort)

- This algorithm process the polygon as if they were being painted into the screen in order of their distance from the viewer.
- Render polygons is back to front order so that polygons behind others are simply painted over



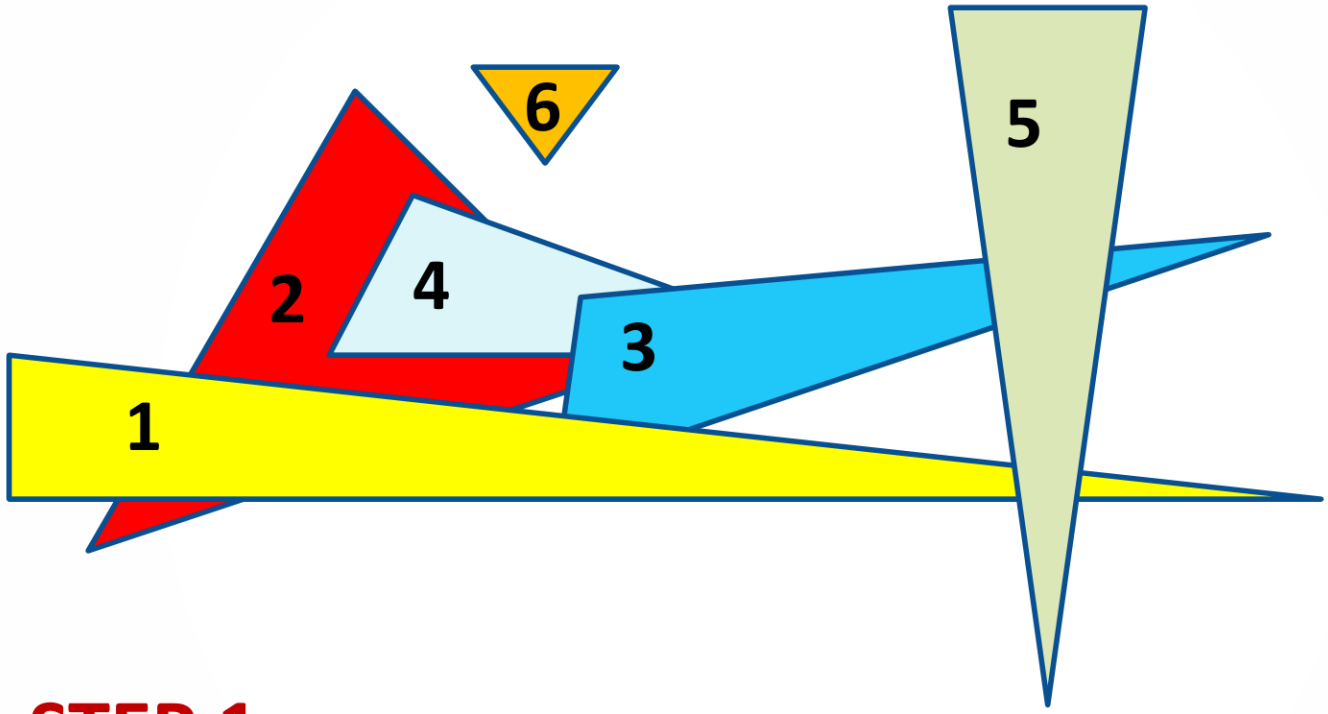
B is behind A as seen by viewer

Fill B then A

The conceptual steps that performed in depth-sort algorithm are

1. Sort all polygon surface according to the smallest (farthest) Z co-ordinate of each.
2. Resolve any ambiguity (doubt) this may cause when the polygons Z extents overlap, splitting polygons if necessary.
3. Scan convert each polygon in ascending order of smaller Z-co-ordinate i.e. farthest surface first (back to front)

The conceptual steps that performed in depth-sort algorithm are

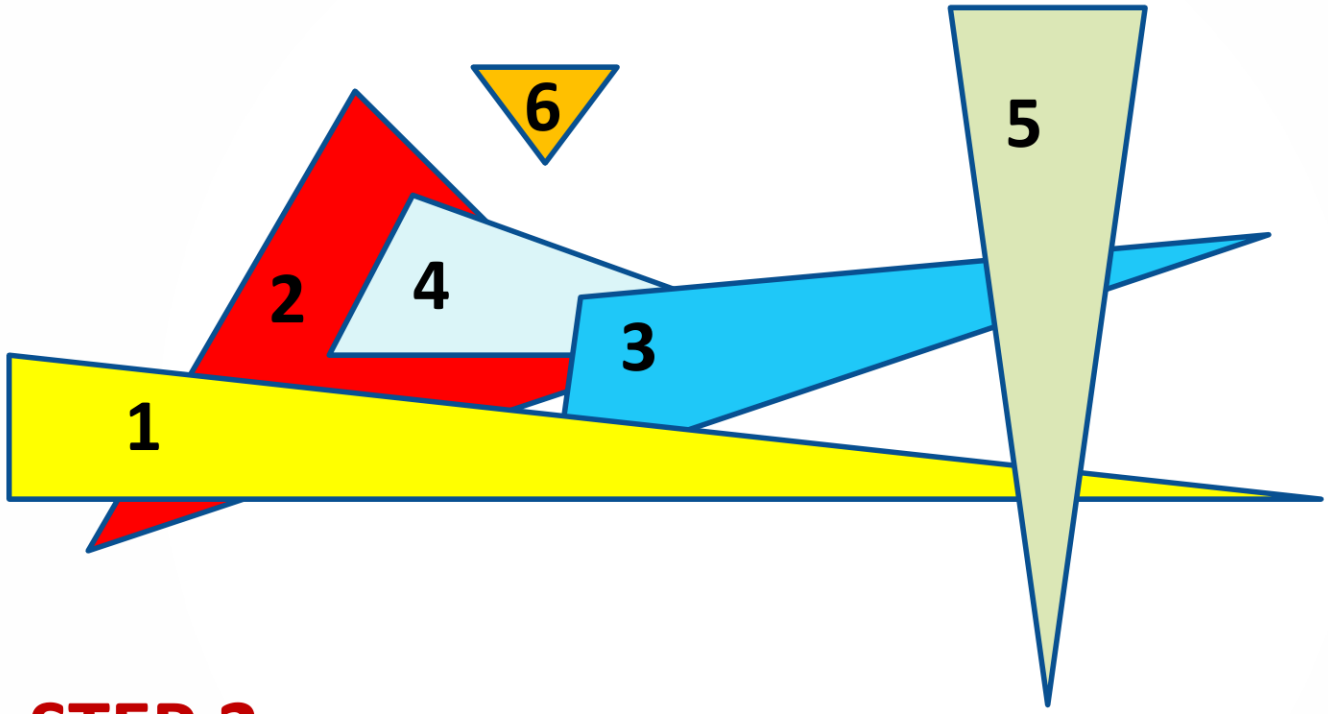


STEP 1

Store the polygon in array by uniquely numbering them

1	2	3	4	5	6

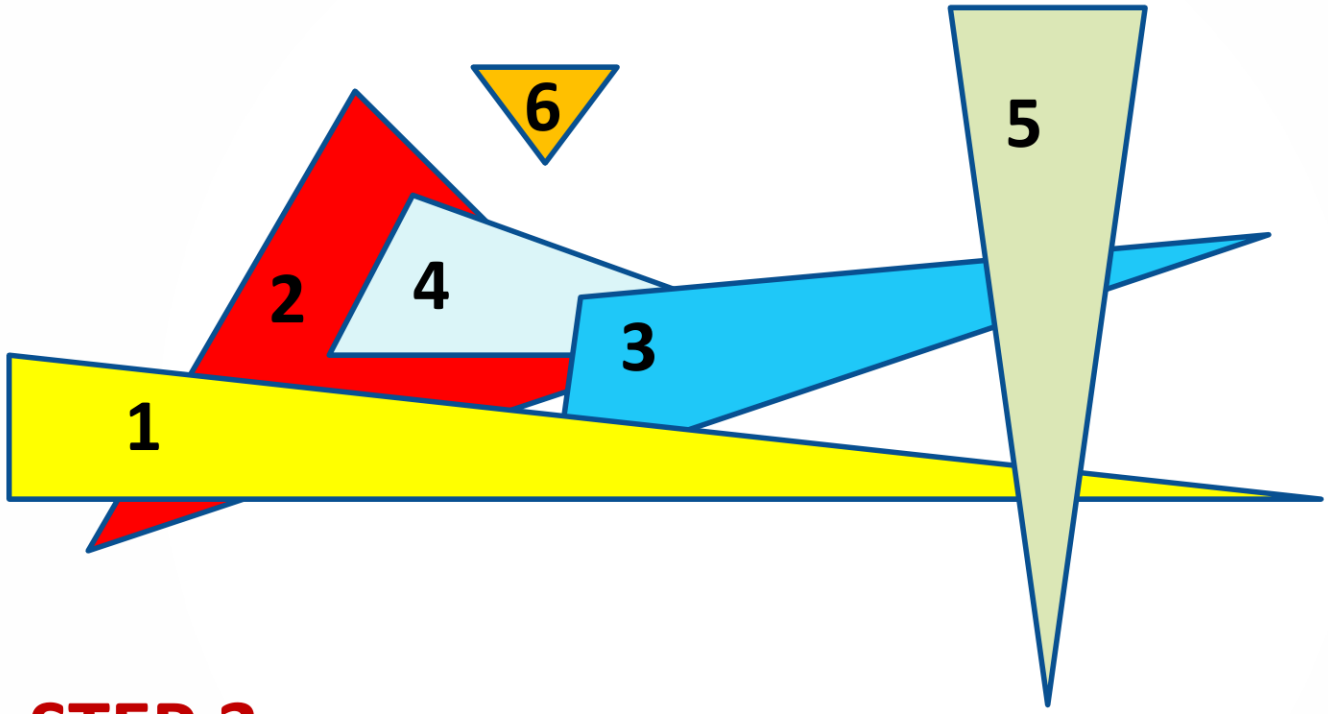
The conceptual steps that performed in depth-sort algorithm are



STEP 2

For each polygon maintain a **linked list** named as **front list** which will contain the polygon no. in front of it in a sequence

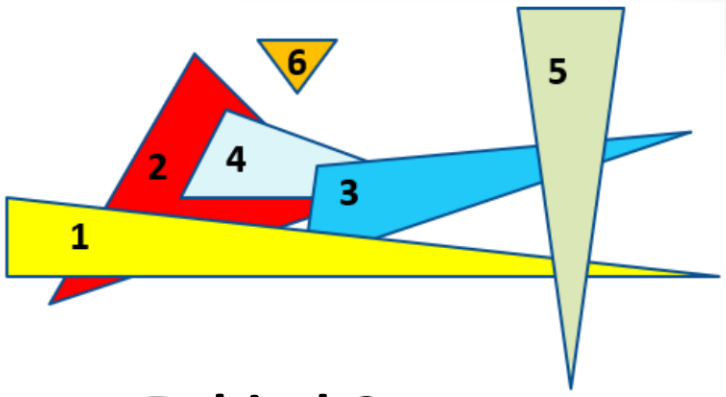
The conceptual steps that performed in depth-sort algorithm are



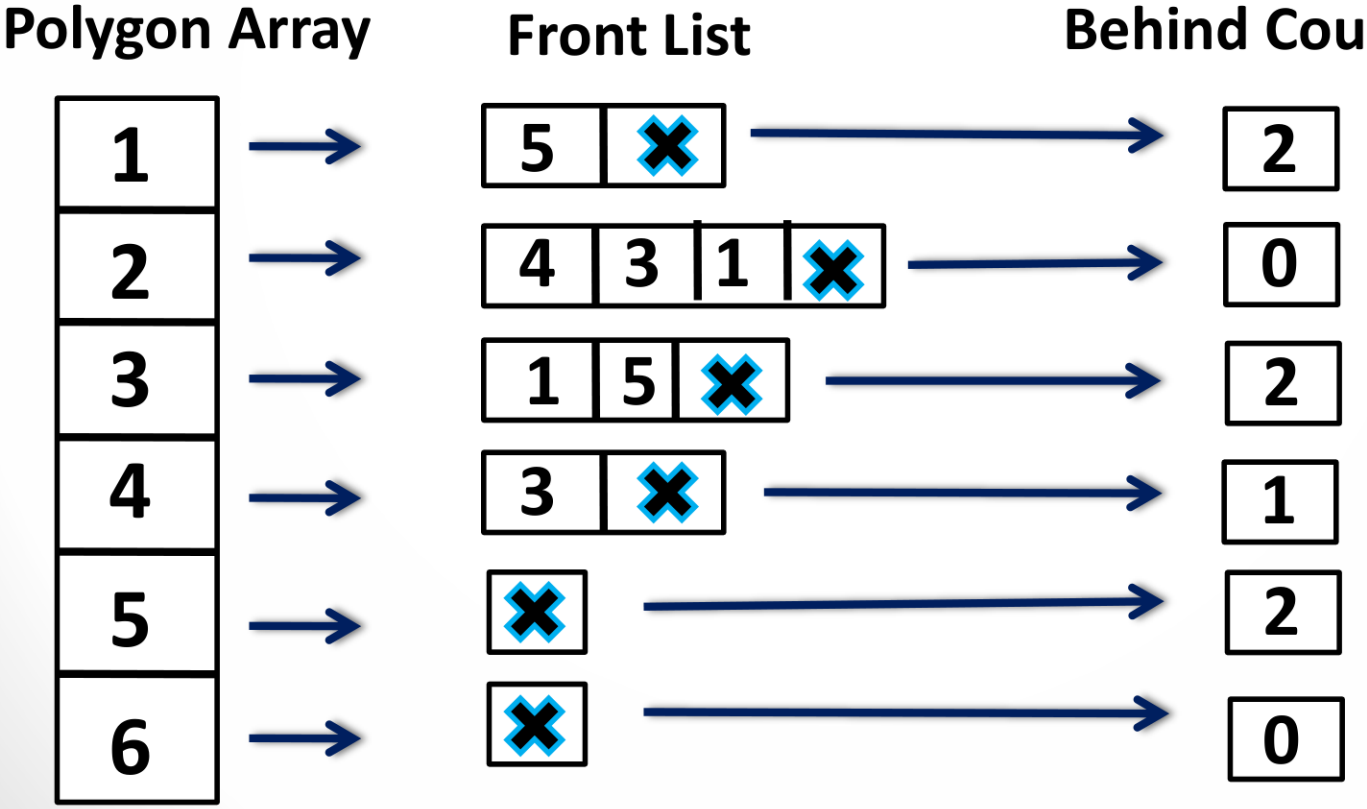
STEP 3

For each polygon maintain a **counter** named as **behind counter** which will contain the no. of polygon behind to it

The conceptual steps that performed in depth-sort algorithm are



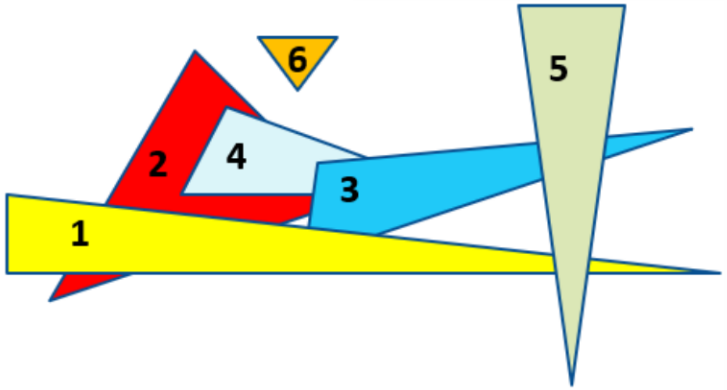
Initialization



The conceptual steps that performed in depth-sort algorithm are

STEP 4 (Loop)

Repeat step 5 & step 6 till all polygon **behind counter is -1**



STEP 5

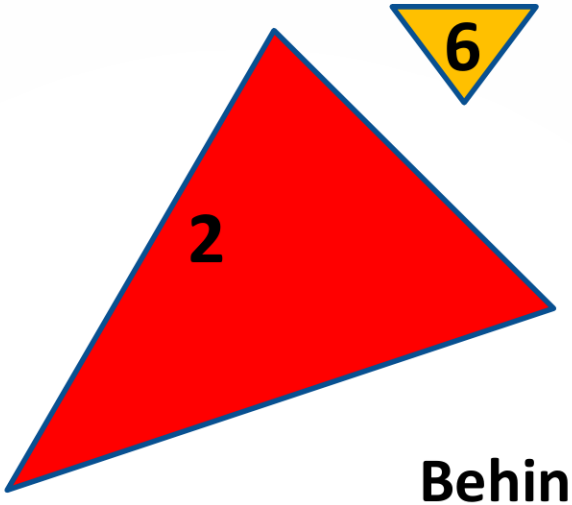
Draw all polygon's whose behind counter is 0

STEP 6

After drawing

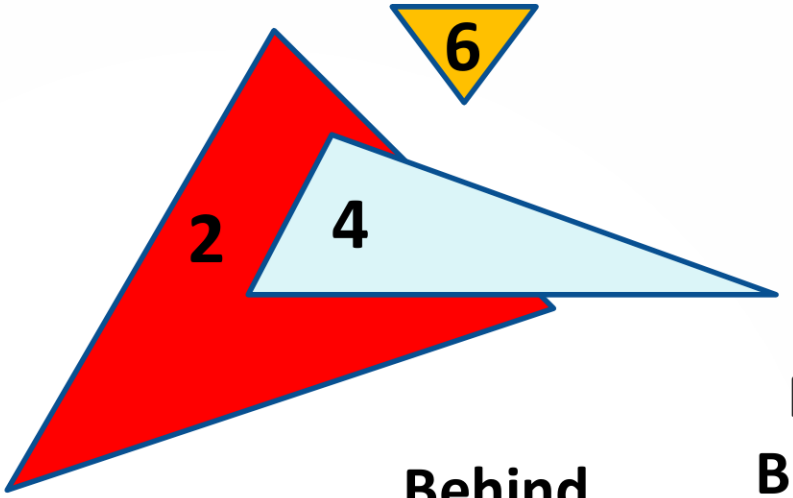
a) Step through the polygon in the front list, decrease their behind counter by 1.
b) For drawn polygon make it's behind counter as **-1**

ITERATION - 1



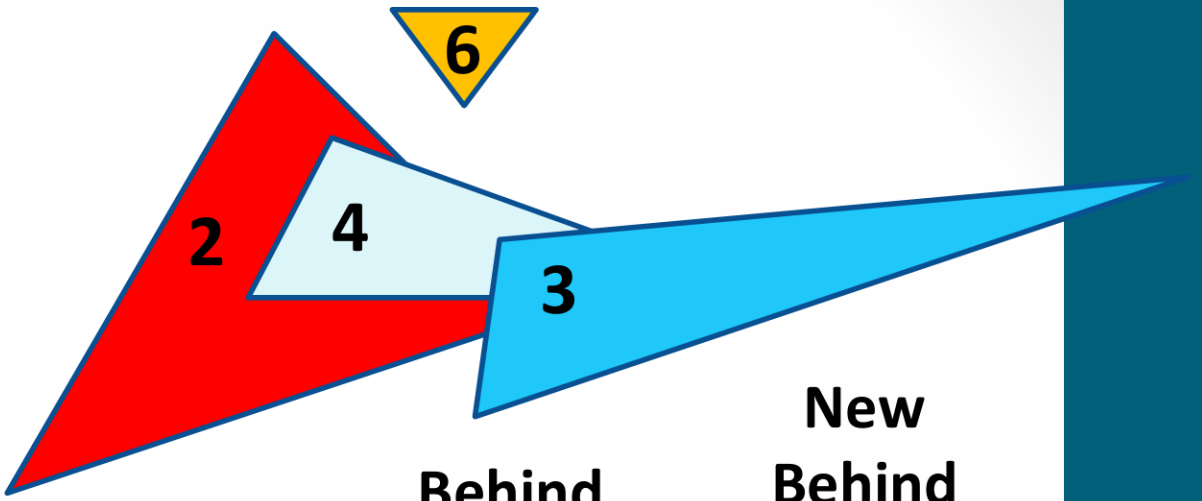
Polygon Array	Front List	Behind Counter	New Behind Counter
1	5 ✖	2	1
2	4 3 1 ✖	0	-1
3	1 5 ✖	2	1
4	3 ✖	1	0
5	✖	2	2
6	✖	0	-1

ITERATION - 2



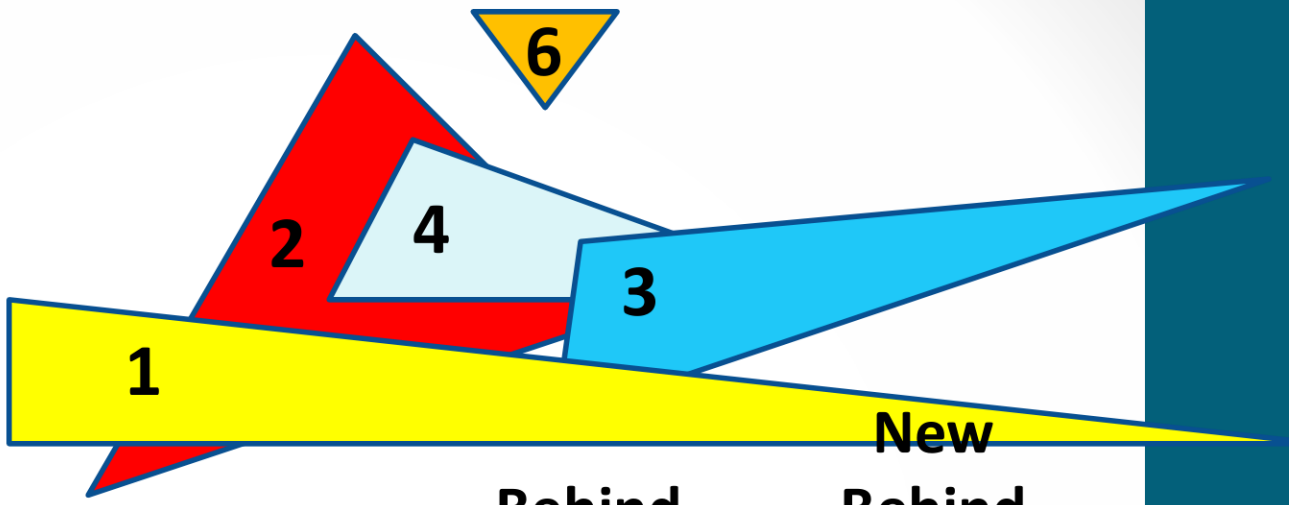
Polygon Array	Front List	Behind Counter	New Behind Counter
1	5 ✖	1	1
2	4 3 1 ✖	-1	-1
3	1 5 ✖	1	0
4	3 ✖	0	-1
5	✖	2	2
6	✖	-1	-1

ITERATION - 3



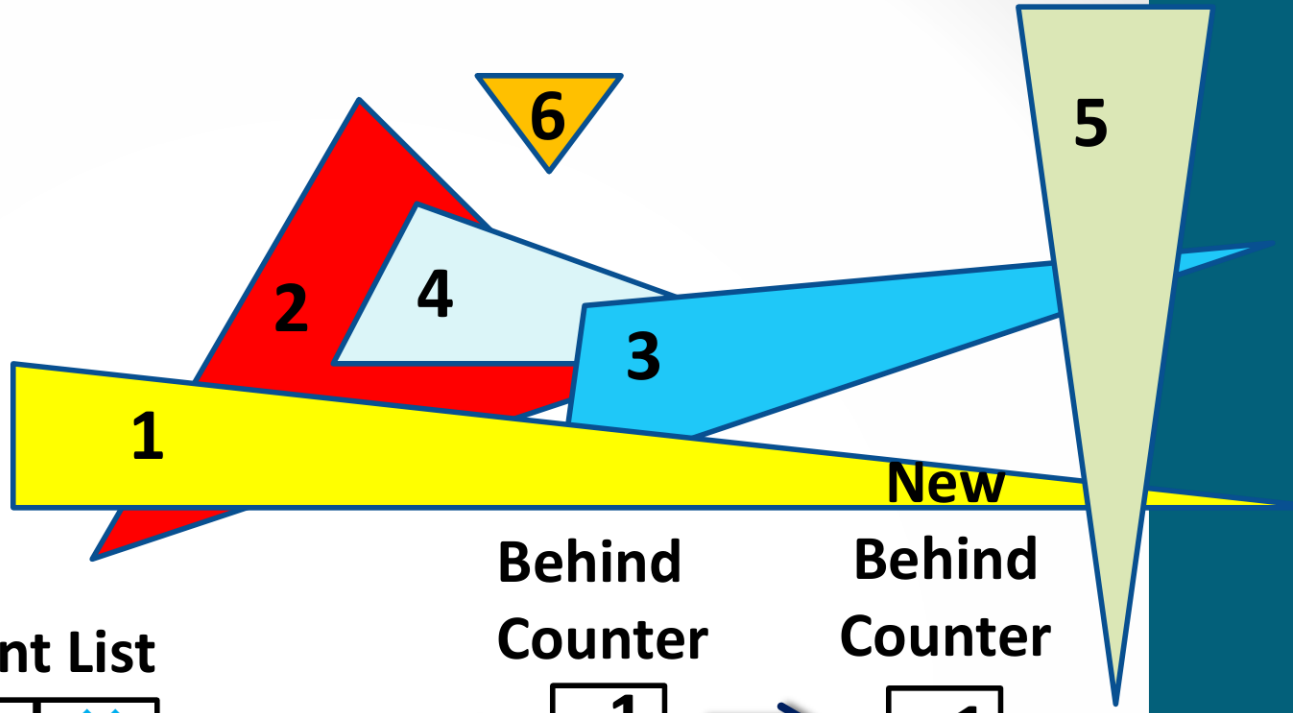
Polygon Array	Front List	Behind Counter	New Behind Counter
1	5 ✖	1	0
2	4 3 1 ✖	-1	-1
3	1 5 ✖	0	-1
4	3 ✖	-1	-1
5	✖	2	1
6	✖	-1	-1

ITERATION - 4



Polygon Array	Front List	Behind Counter	Behind Counter
1	5 ✖	0	-1
2	4 3 1 ✖	-1	-1
3	1 5 ✖	-1	-1
4	3 ✖	-1	-1
5	✖	1	0
6	✖	-1	-1

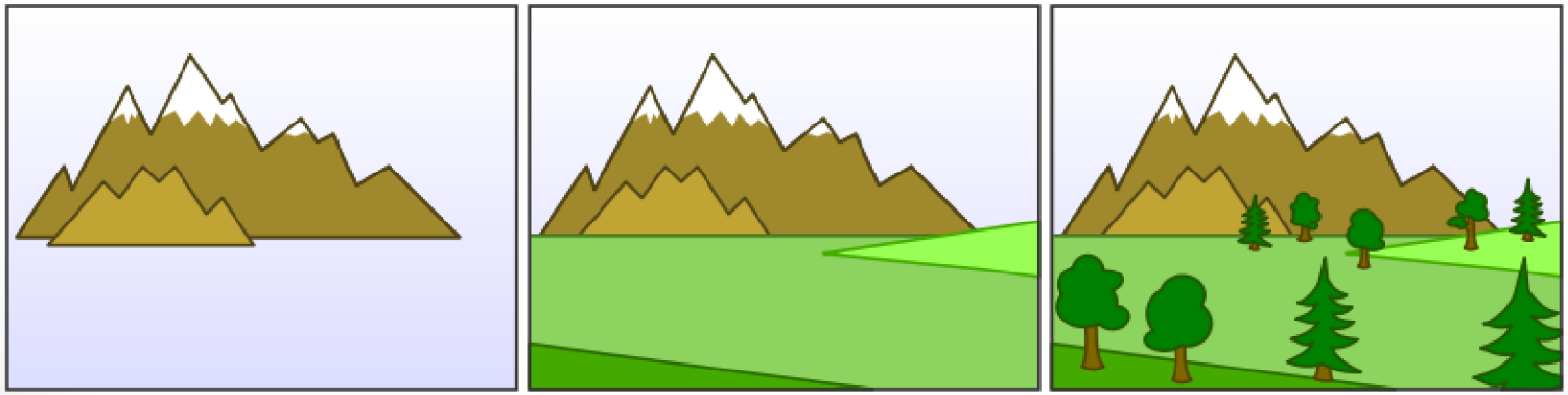
ITERATION - 5



Polygon Array	Front List	Behind Counter	Behind Counter
1	5 ✖	-1	-1
2	4 3 1 ✖	-1	-1
3	1 5 ✖	-1	-1
4	3 ✖	-1	-1
5	✖	0	-1
6	✖	-1	-1

The Painter's Algorithm (Depth Sort)

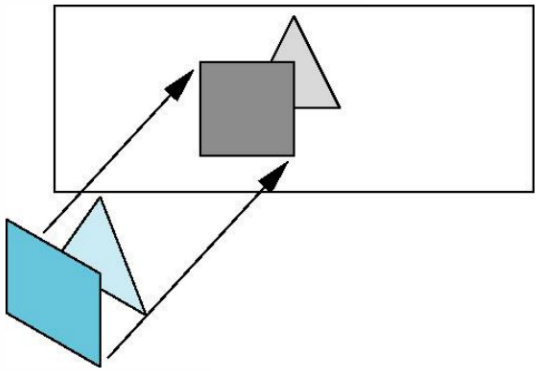
- In this method, the newly displayed surface is partly or completely obscure the previously displayed surface.
- Essentially, we are sorting the surface into priority order such that surface with lower priority (lower z, far objects) can be obscured by those with higher priority (high z-value).



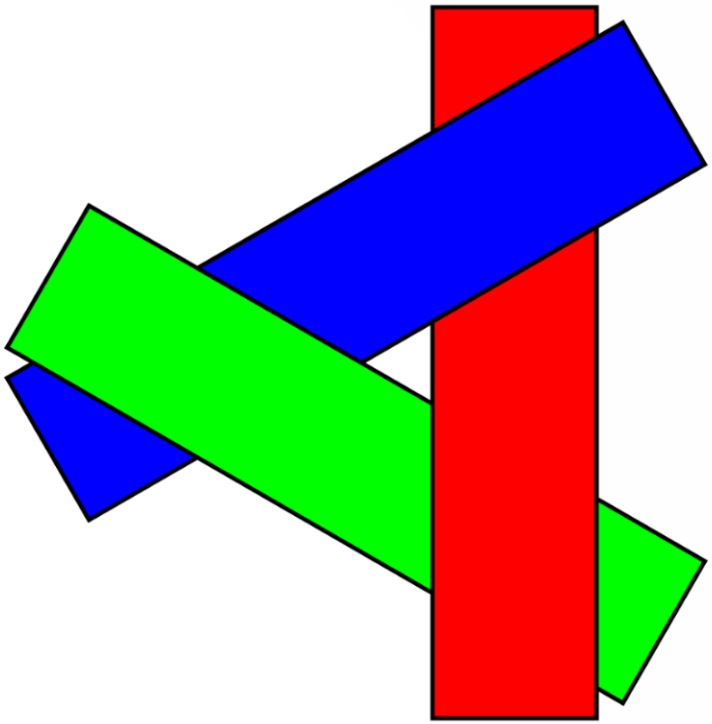
The Painter's Algorithm (Depth Sort)

- This algorithm is also called "***Painter's Algorithm***" as it simulates how a painter typically produces his painting by starting with the background and then progressively adding new (nearer) objects to the canvas.
- Thus, each layer of paint covers up the previous layer.
- Similarly, we first sort surfaces according to their distance from the view plane.
- The intensity values for the farthest surface are then entered into the refresh buffer.
- Taking each succeeding surface in turn (in decreasing depth order), we "***paint***" the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.

The Painter's Algorithm (Depth Sort)



Easy Cases:
One polygon is completely behind the other. Just "*paint*" over it.



Problem
Cyclic Overlap:
i.e. intersecting polygon surfaces

The Painter's Algorithm (Depth Sort)

Solution

- For intersecting polygons, we can split one polygon into two or more polygons which can then be painted from back to front.
- This needs more time to compute intersection between polygons.
- So it becomes complex algorithm for such surface existence.

The Painter's Algorithm (Depth Sort)

Step 1 : Sort all the polygons in the decreasing order of their depth values.

Step 2 : Determine all the polygons Q(preceding P) in the polygon list whose z extents overlap that of P.

Step 3 : Perform test 2 to 6, for each Q in the list until true.

(a) If every Q passes the test, then paint the polygon P.

(b) If the test fails for some Q, swap P and Q in the list, and make some indication that Q is swapped. If Q is already swapped, use the plane containing polygon P to divide polygon Q in to two polygons, Q_1 and Q_2. Replace Q with Q_1 and Q_2. Repeat step 3.

Step 4 : Stop.

The Painter's Algorithm (Depth Sort)

Advantages

+ no extra storage required

+ no per-pixel operations required

Disadvantages

- if polygons cannot be ordered they have to be split

- sorting is expensive; when the viewpoint changes, objects have to be sorted again.

Scan-Line Method

- This ***image-space*** method for removing hidden surfaces is an extension of the ***scan-line algorithm*** for filling polygon interiors where, we deal with multiple surfaces rather than one.
- Each scan line is processed with calculating the depth for nearest view for determining the visible surface of intersecting polygon.
- When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.

Scan-Line Method

- In this method, as each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible.
- Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane.
- When the visible surface has been determined, the intensity value for that position is entered into the image buffer

Scan-Line Method

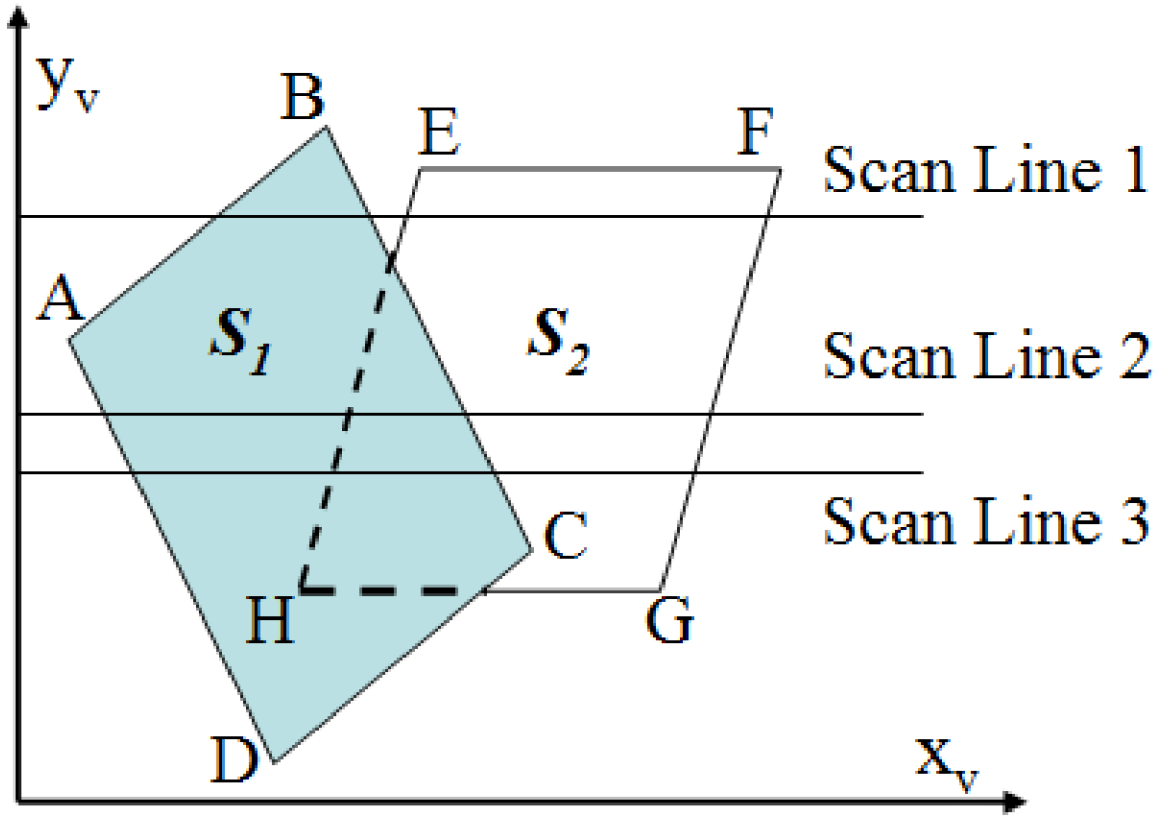


Fig. Scan lines crossing the projection of two surfaces, S_1 and S_2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

Scan-Line Method

- To facilitate the search for surfaces crossing a given scan line, we can set up an ***active list*** of edges from information in the **edge table** that contain only edges that cross the current scan line, sorted in order of increasing ***x.***
- In addition, we define a ***flag*** for each surface that is set **on** or **off** to indicate whether a position along a scan line is inside or outside of the surface.
- Scan lines are processed from left to right.
- At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

Scan-Line Method

DATA STRUCTURE

A. Edge table containing
- Coordinate endpoints for each line in a scene
- Inverse slope of each line
- Pointers into polygon table to identify the surfaces bounded by each line

B. Surface table containing
- Coefficients of the plane equation for each surface
- Intensity information for each surface
- Pointers to edge table

C. Active Edge List
- To keep a trace of which edges are intersected by the given scan line

Note:

❑The edges are sorted in order of increasing x

❑Define flags for each surface to indicate whether a position is inside or outside the surface

Scan-Line Method

I. Initialize the necessary data structure

1. Edge table containing end point coordinates, inverse slope and polygon pointer.
2. Surface table containing plane coefficients and surface intensity
3. Active Edge List
4. Flag for each surface

II. For each scan line repeat

1. update active edge list
2. determine point of intersection and set surface on or off.
3. If flag is on, store its value in the refresh buffer
4. If more than one surface is on, do depth sorting and store the intensity of surface nearest to view plane in the refresh buffer

Scan-Line Algorithm

(1) Sort polygons into sorted surface table (SST) based on Y

(2) Initialize y and active surface table (AST)

y = first nonempty scanline

AST = SST [y]

(3) Repeat until AST and SST are empty

identify spans for this scanline (sorted on x)

For each span

determine visible element (based on z)

fill pixel intensities with values from visible element

Update AST

remove exhausted polygons

y++

update x intercepts

resort AST on x

add entering polygons

(4) Display Intensity Array

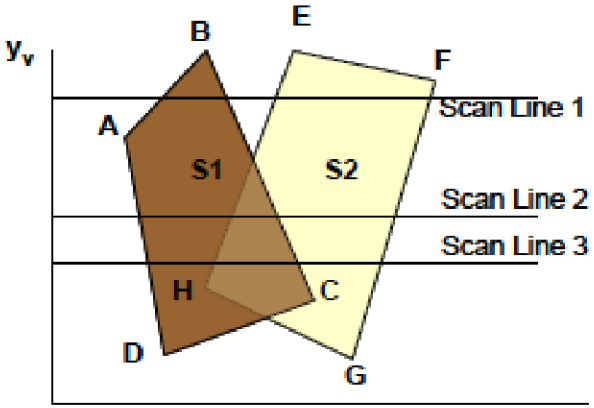
Scan-Line Method

```
For each scan line do
    Begin
        For each pixel (x,y) along the scan line do          ---------- Step 1
        Begin
            z_buffer(x,y) = 0
            Image_buffer(x,y) = background_color
        End

        For each polygon in the scene do                     ---------- Step 2
        Begin
            For each pixel (x,y) along the scan line that is covered by the polygon do
            Begin
                2a. Compute the depth or z of the polygon at pixel location (x,y).
                2b. If z < z_buffer(x,y) then
                        Set z_buffer(x,y) = z
                        Set Image_buffer(x,y) = polygon's colour
            End
        End
    End
```

Scan-Line Method

- **For scan line 1**
  - The active edge list contains edges AB,BC,EH, FG
  - Between edges AB and BC, only *flags for s1 == on* and between edges EH and FG, only *flags for s2==on*
    - no depth calculation needed and corresponding surface intensities are entered in refresh buffer

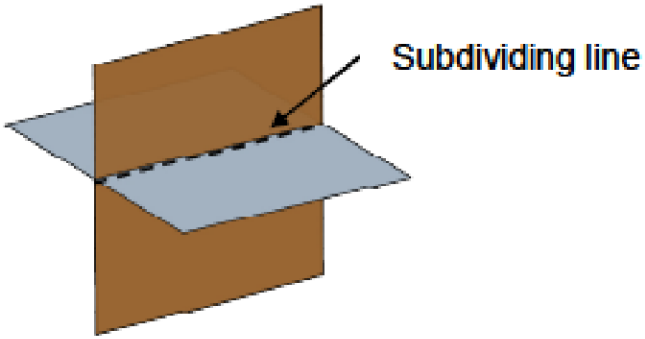


- **For scan line 2**
  - The active edge list contains edges AD,EH,BC and FG
  - Between edges AD and EH, only the *flag for surface s1 == on*
  - Between edges EH and BC *flags for both surfaces == on*
    - Depth calculation (using plane coefficients) is needed.
  - In this example ,say s2 is nearer to the view plane than s1, so intensities for surface s2 are loaded into the refresh buffer until boundary BC is encountered
  - Between edges BC and FG flag for s1==off and *flag for s2 == on*
  - Intensities for s2 are loaded on refresh buffer
- **For scan line 3**
  - Same **coherent** property as scan line 2 as noticed from active list, so no depth

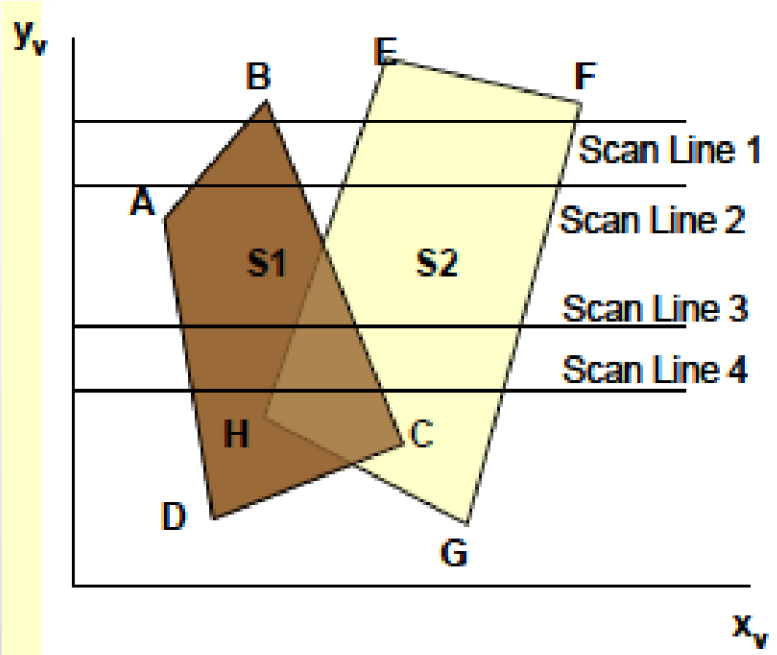
Scan-Line Method

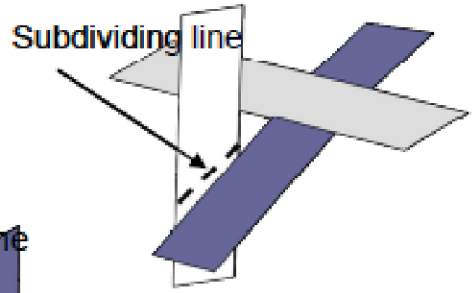
Problem:

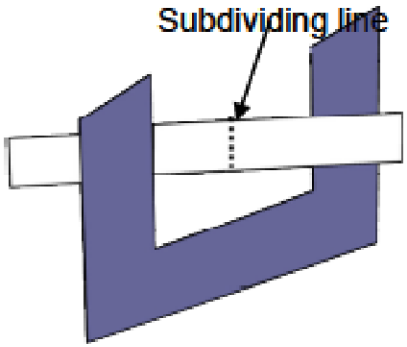
Dealing with **cut through** surfaces and **cyclic overlap** is problematic when used coherent properties



Solution: Divide the surface to eliminate the overlap or cut through







Scan-Line Algorithm

Advantages

+ little memory required
+ can generate scan lines as required
+ can anti-alias within scan line fast
+ simplification of problem simplifies geometry
+ can exploit coherence

Disadvantages

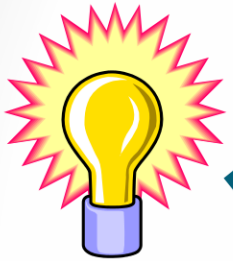
- fairly complicated to implement
- difficult to anti-alias between scanlines

Assignment

1. What do you mean by visible surface detection? Why visible surface should be detected?
2. What do you understand by hidden surface removal technique?
3. Explain Z buffer algorithm for visible surface detection. How does the Z-buffer algorithm determine which surfaces are hidden.
4. Explain the scan line algorithm for visible surface detection.
5. Write the difference between object space method and image space method.
6. Explain Painter's algorithm for visible surface detection.
7. Compare scan-line and Painter's algorithm method.
8. Why depth sorting method is said to be Painter's algorithm? Explain depth sorting algorithm.
9. Write Short notes on:
    a. Back-face detection method

Light

Source emits photons

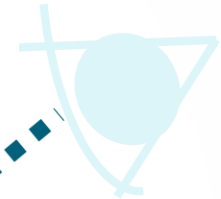


Photons travel in a straight line

When they hit an object they:

- ❑ bounce off in a new direction
- ❑ are absorbed

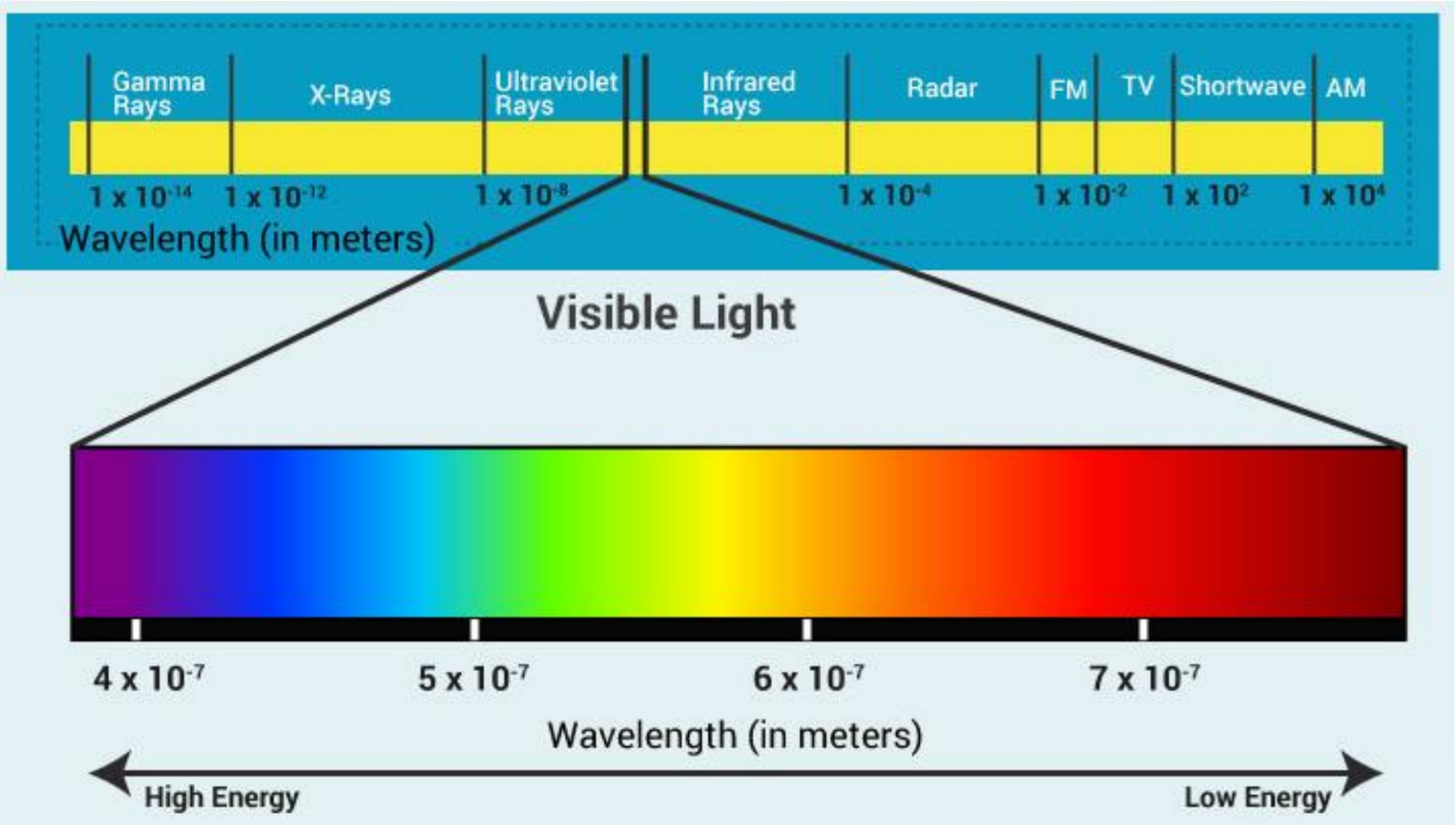
And then some reach the eye/camera.



Light

- Light or visible light is electromagnetic radiation that can be perceived by the human eye.
- Light is an electromagnetic wave.
- Visible light is the portion of the electromagnetic spectrum.
- The broad range of wavelengths is known as the *electromagnetic spectrum.*
- Visible light is a form of electromagnetic (EM) radiation, as are radio waves, infrared radiation, ultraviolet radiation, X-rays and microwaves.

Light


Gamma Rays
X-Rays
Ultraviolet Rays
Infrared Rays
Radar
FM
TV
Shortwave
AM
1 x 10-14
1 x 10-12
1 x 10-8
1 x 10-4
1 x 10-2
1 x 102
1 x 104
Wavelength (in meters)
Visible Light
4 x 10-7
5 x 10-7
6 x 10-7
7 x 10-7
Wavelength (in meters)
High Energy
Low Energy

Light

- Electromagnetic spectrum is typically divided into seven regions in order of decreasing wavelength and increasing energy and frequency.
- These regions are:
  - radio waves
  - microwaves
  - infrared (IR)
  - visible light
  - ultraviolet (UV)
  - X-rays
  - gamma-rays

Light

- Radio waves (wavelengths greater than 0.4 inch, or 10 millimeters)
- Microwaves (wavelengths between 0.004 and 0.4 inch, or 0.1 to 10 mm)
- Infrared (IR) (wavelengths between 0.00003 and 0.004 inch, or 740 nanometers to 100 micrometers)
- Visible light (wavelengths between 0.000015 and 0.00003 inch, or 380 to 740 nanometers)
- Ultraviolet (UV) (wavelengths between 0.000015 and 0.00003 inch, or 380 to 740 nanometers)
- X-rays (wavelengths between 4×10^{-7} to 4×10^{-8} inch, or 100 picometers to 10 nanometers)
- Gamma-rays (wavelengths less than 4×10^{-9} inch, or 100 picometers)

Light

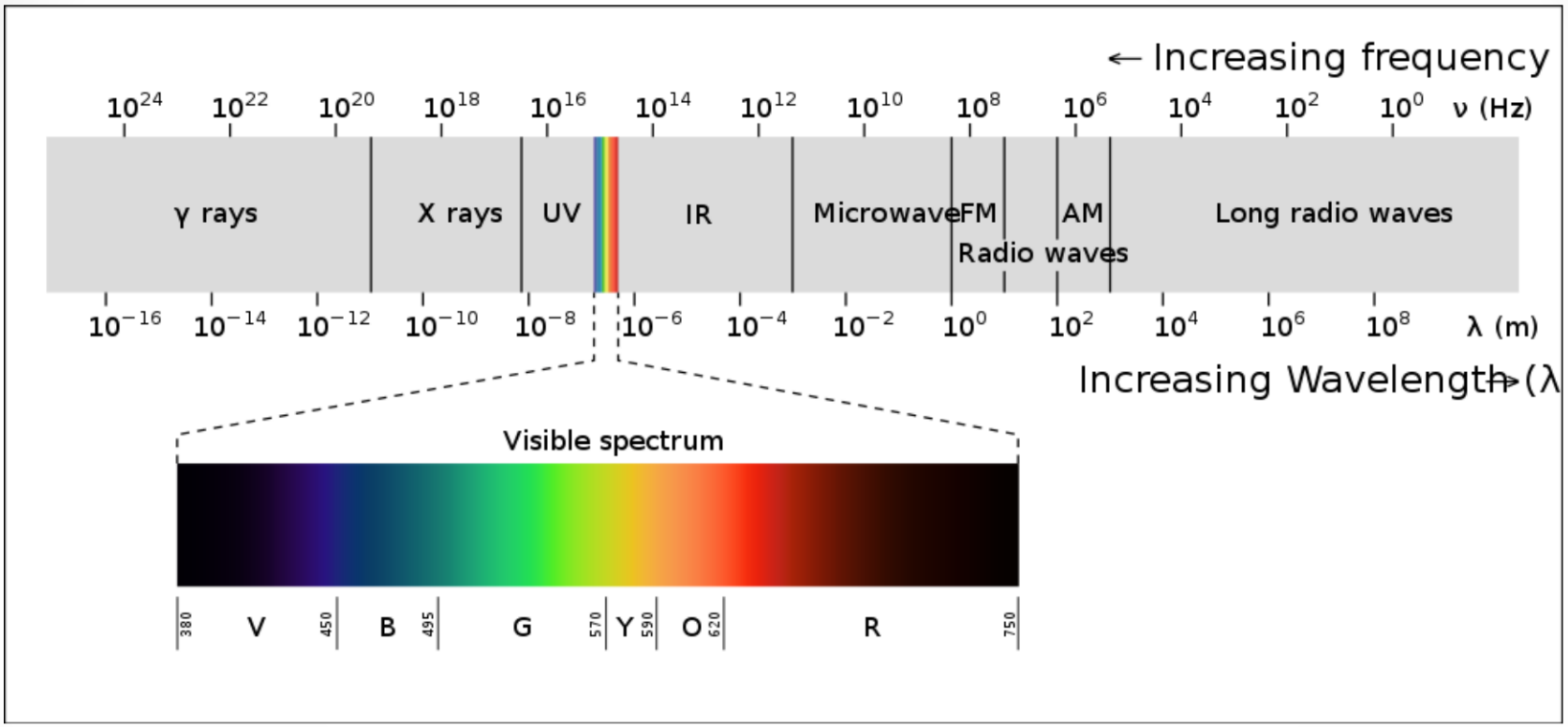


Figure: The electromagnetic spectrum, with the visible portion highlighted.

Light

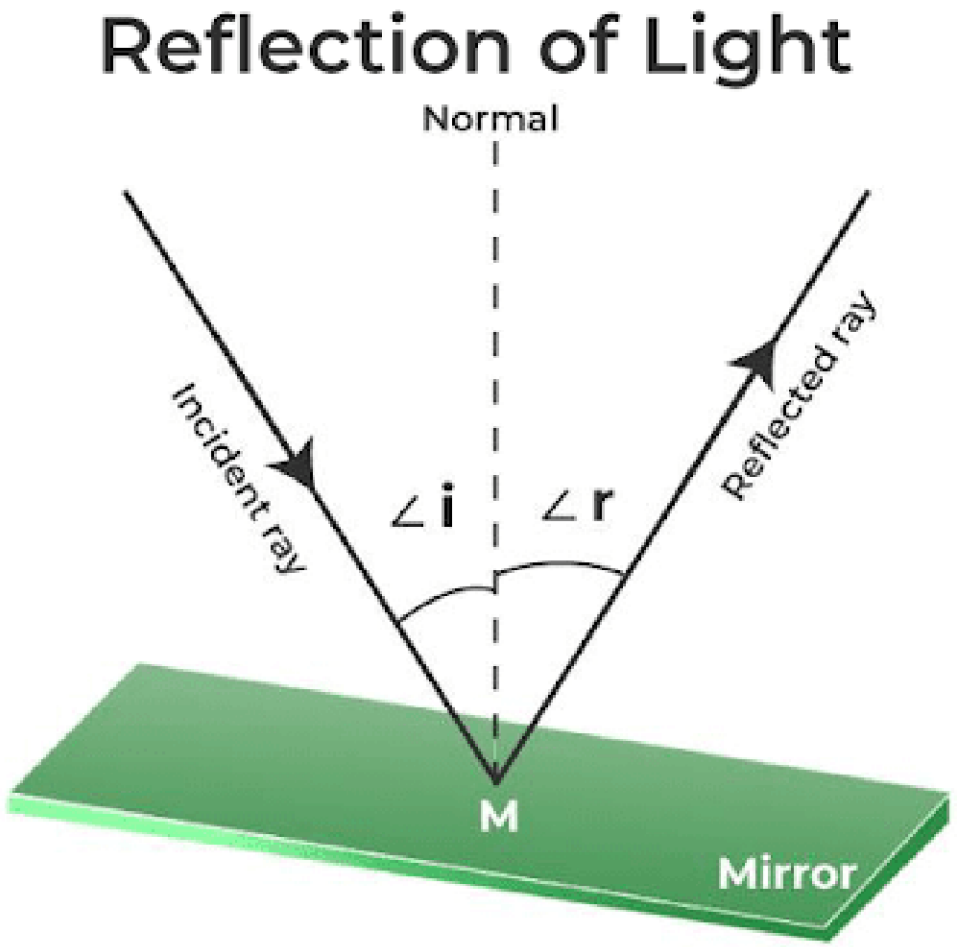
	Wavelength	Frequenzy
Red	~ 625 – 740 nm	~ 480 – 405 THz
Orange	~ 590 – 625 nm	~ 510 – 480 THz
Yellow	~ 565 – 590 nm	~ 530 – 510 THz
Green	~ 520 – 565 nm	~ 580 – 530 THz
Blue	~ 445 – 520 nm	~ 675 – 580 THz
Indigo	~ 425 – 445 nm	~ 700 – 675 THz
Violet	~ 380 – 425 nm	~ 790 – 700 THz

Table: The approximate wavelength and frequency of visible light

Properties of Light

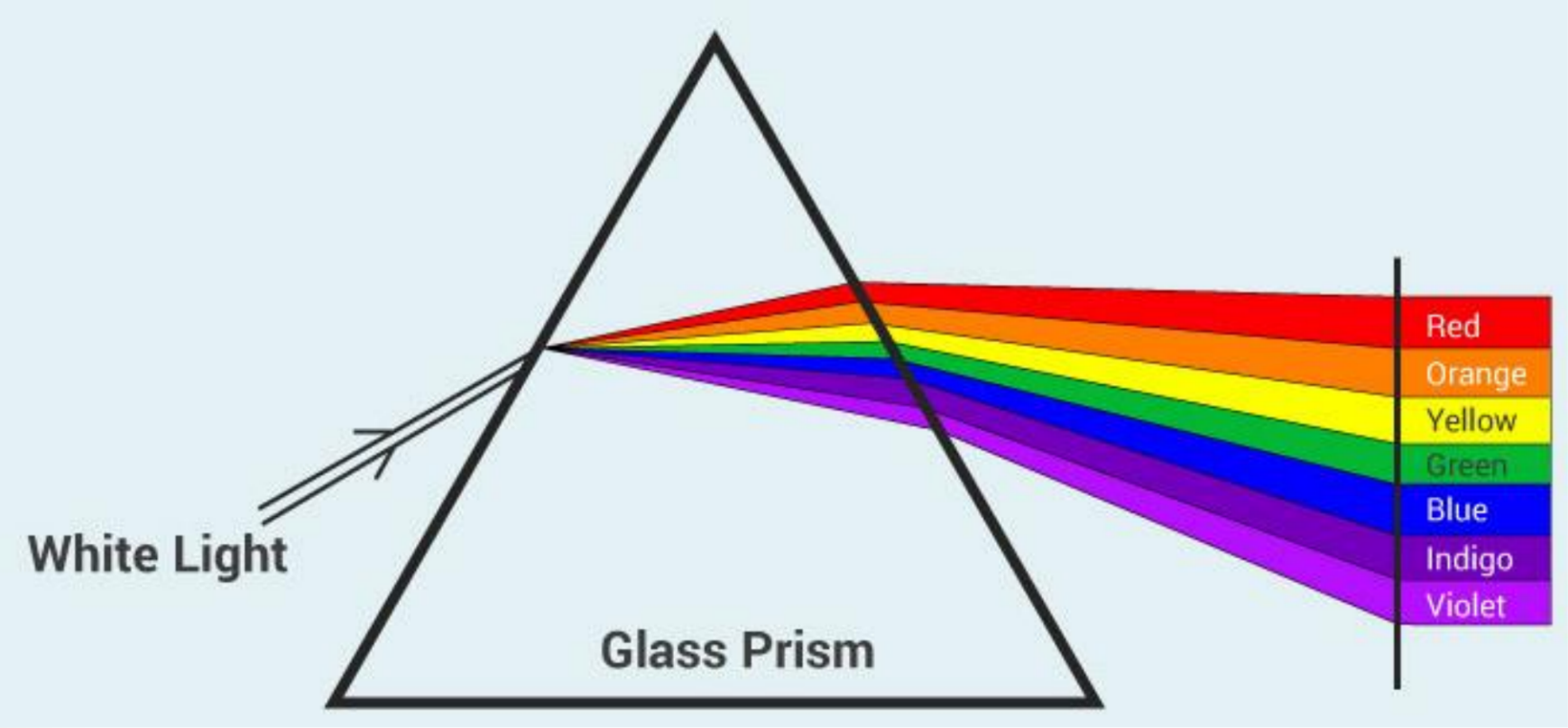
- The primary properties of light are reflection, refraction, dispersion, diffraction, polarization, interference, intensity, propagation direction, frequency or wavelength spectrum, etc.
- Light travels in a straight line.
- The speed of light is faster than sound. Light travels at a speed of 3 x 108 m/s.
- Visible light falls in the range of the EM spectrum between infrared (IR) and ultraviolet (UV).
- It has frequencies of about 4×10^{14} to 8×10^{14} cycles per second, or hertz (Hz).
- It has wavelengths of about 740 nanometers (nm) or 2.9×10^{-5} inches, to 380 nm (1.5×10^{-5} inches).
- Light is not composed of RGB
- Light is linear.

Properties of Light



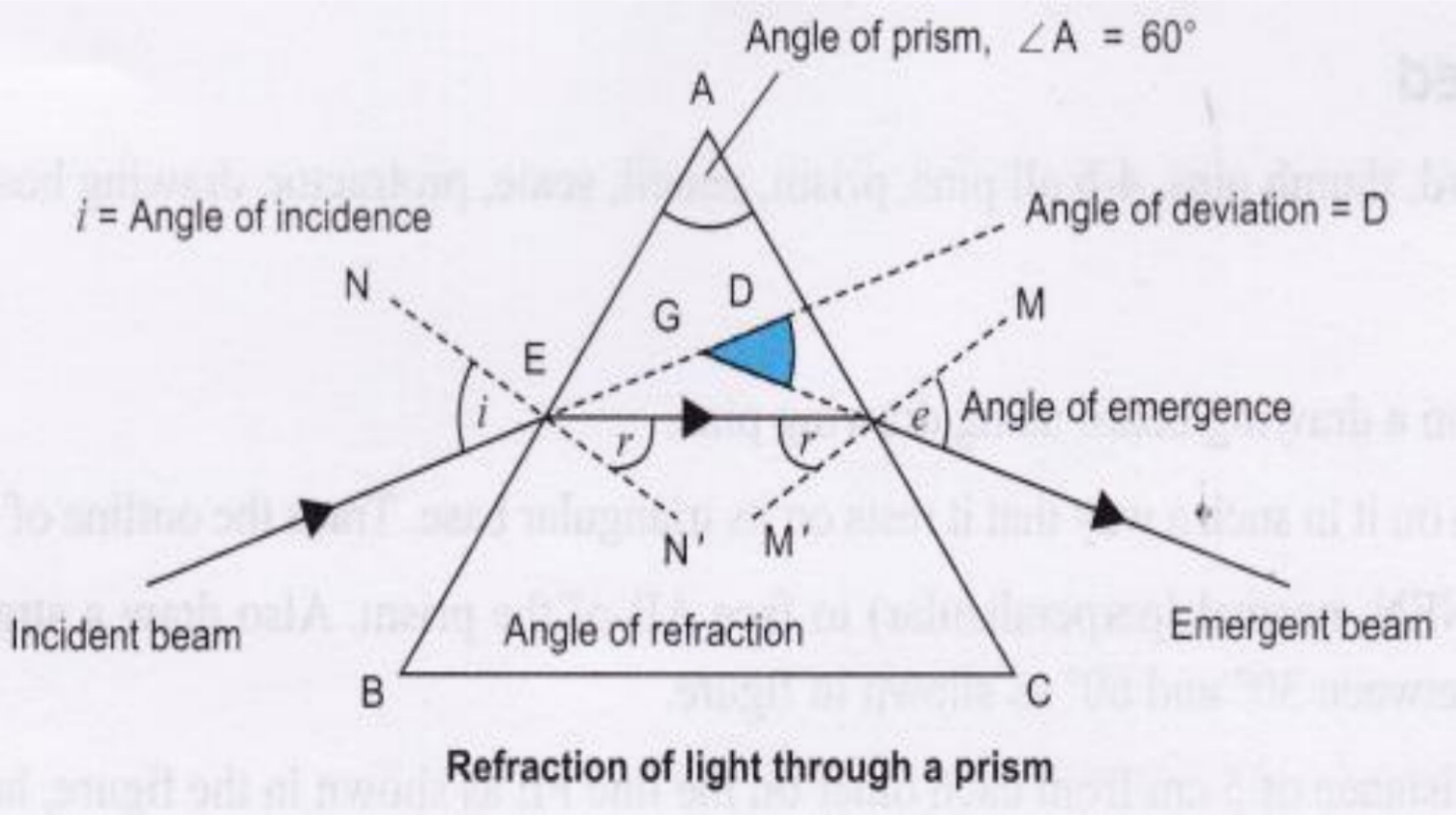
Reflection is the phenomenon in which light travelling in one medium, incident on the surface of another returns to the first medium, obeying the laws of reflection.

Properties of Light



The splitting up of white light into seven colors on passing through a transparent medium like a glass prism is called ***dispersion*** of light.

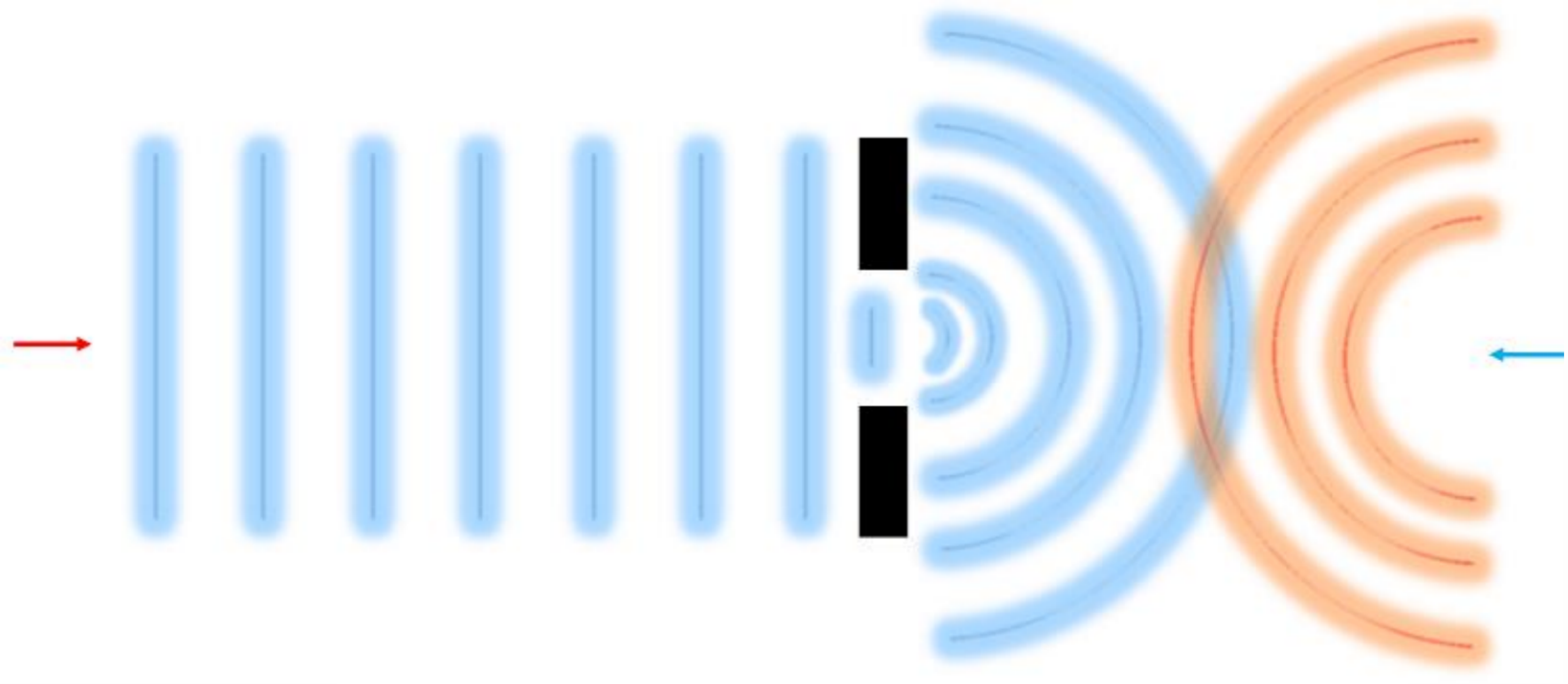
Properties of Light



Refraction of light through a prism

The direction of light changes as it passes from one medium to another. In other words, light bends as it travels from one medium to another. This phenomenon is known as ***refraction***.

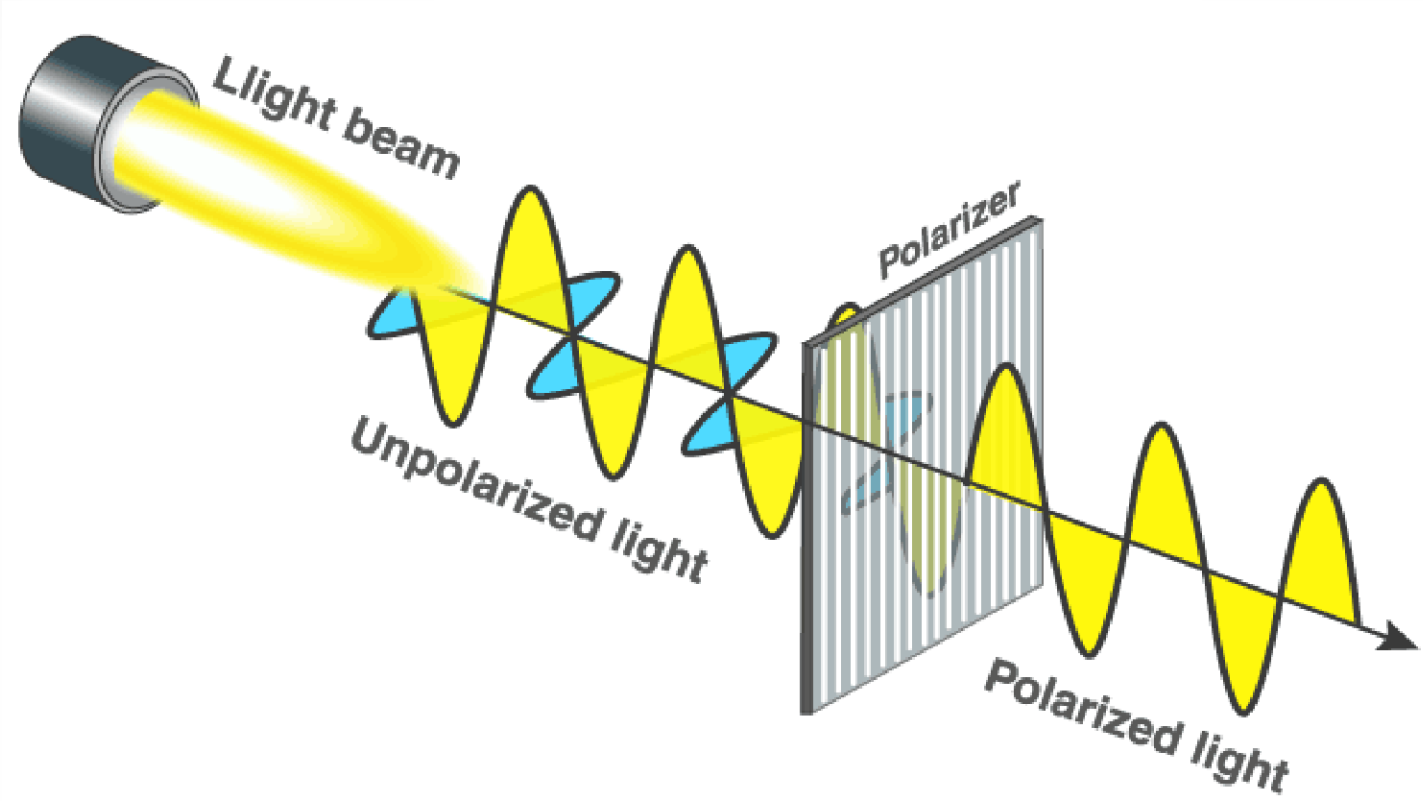
Properties of Light



The phenomenon of bending of light around corners of small obstacles and hence it's encroachment into the region of the geometrical shadow is called ***diffraction***.

Interference is the phenomenon of modification in the intensity of light due to redistribution of light energy in the region of superposition of two or more light waves.

Properties of Light



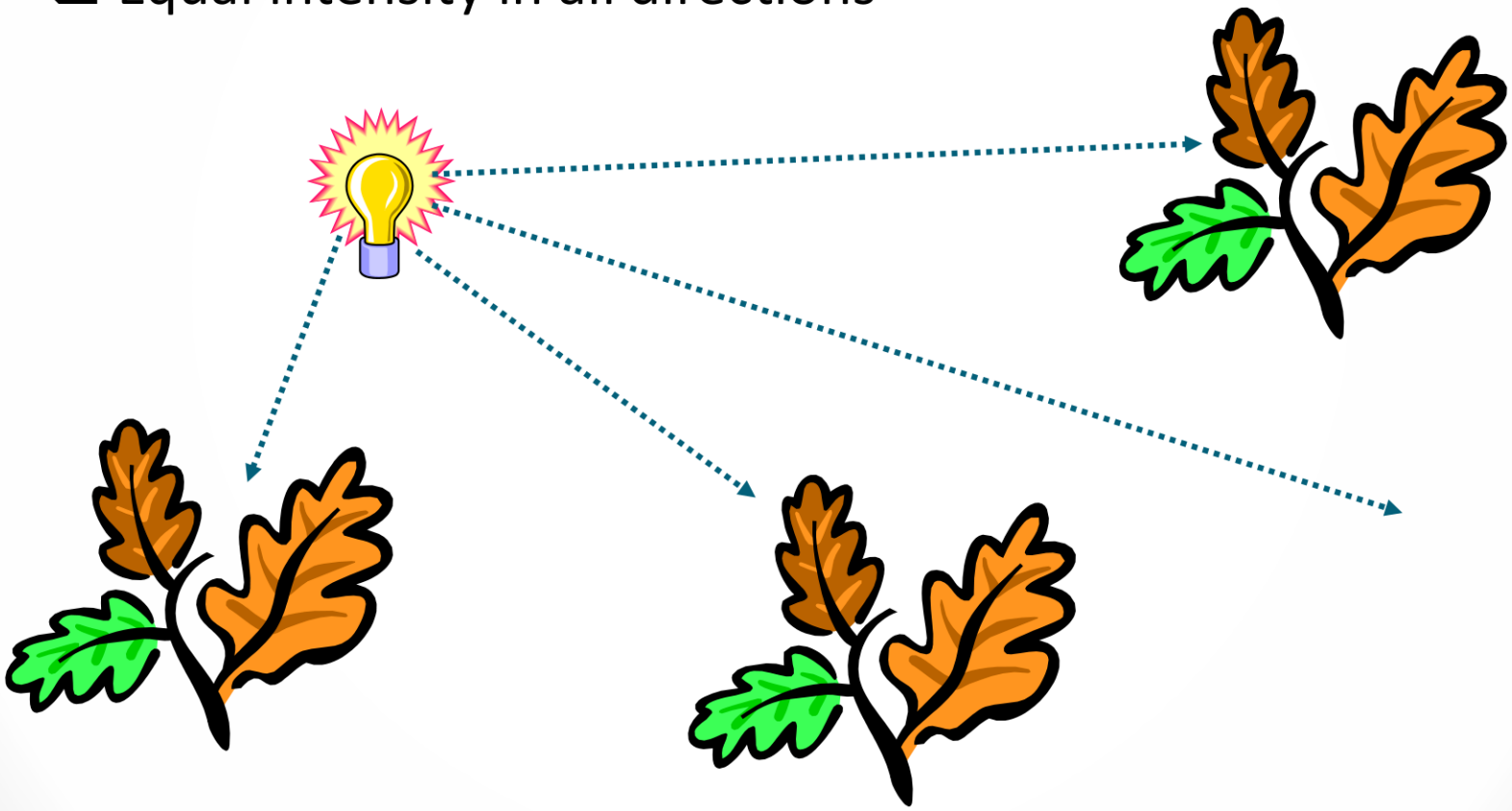
Normal light vibrates in all directions perpendicular to the propagation of light. If the light is constrained to vibrate in only one particular plane, then the light is called polarised light. The phenomenon is called ***polarisation***.

Light Sources

- Sometimes light sources are referred as *light emitting object* and *light reflectors*.
- Generally light source is used to mean an object that is emitting radiant energy. Example: Sun.
- **Point Source:** Point source is the simplest light emitter e.g. light bulb.
- **Distributed Light source:** e.g. Fluorescent light

Light Sources: Point Source

- Light emanates from a point
- Equal intensity in all directions



Light Sources

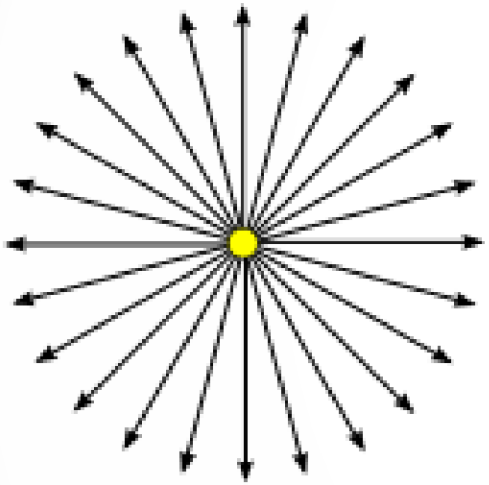


Figure: Diverging ray paths from the Point light source

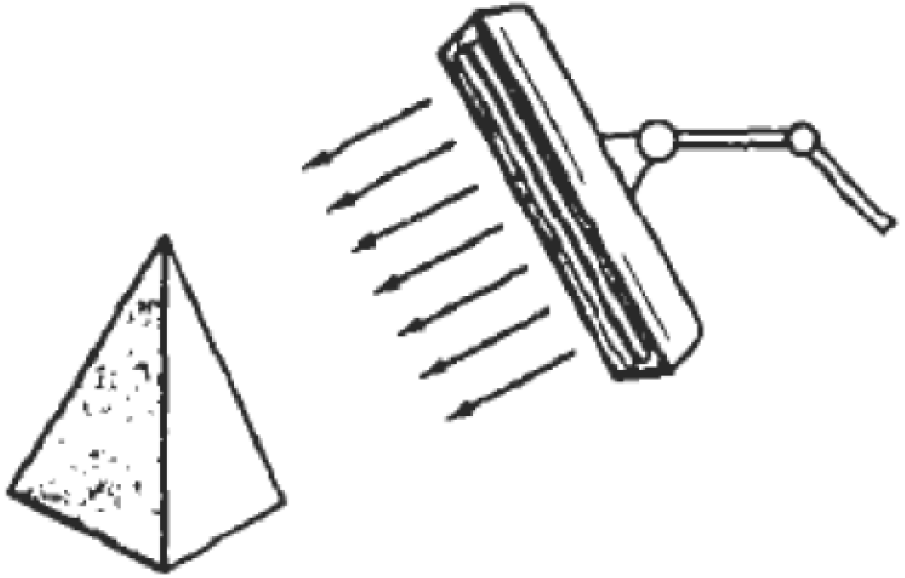
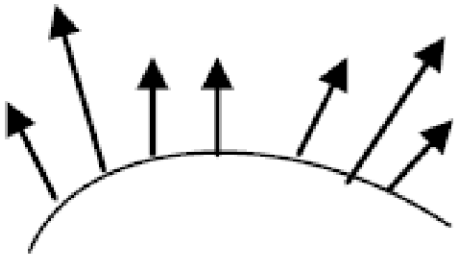


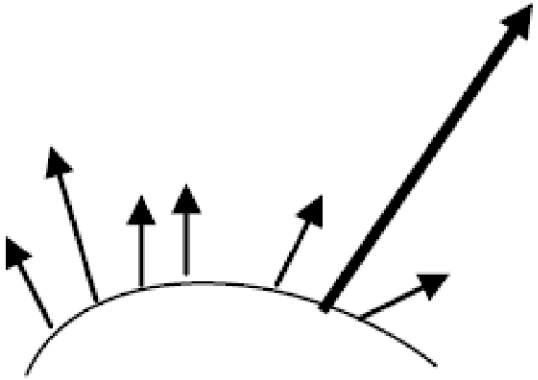
Fig: An object illuminated with a distributed light source

Light Sources

- When light is incident on an opaque surface part of it is *reflected* and part of it is *absorbed*.
- Surface that are rough or grainy, tend to scatter the reflected light in all direction which is called *diffuse reflection*.
- When light sources create highlights, or bright spots, called *specular reflection*



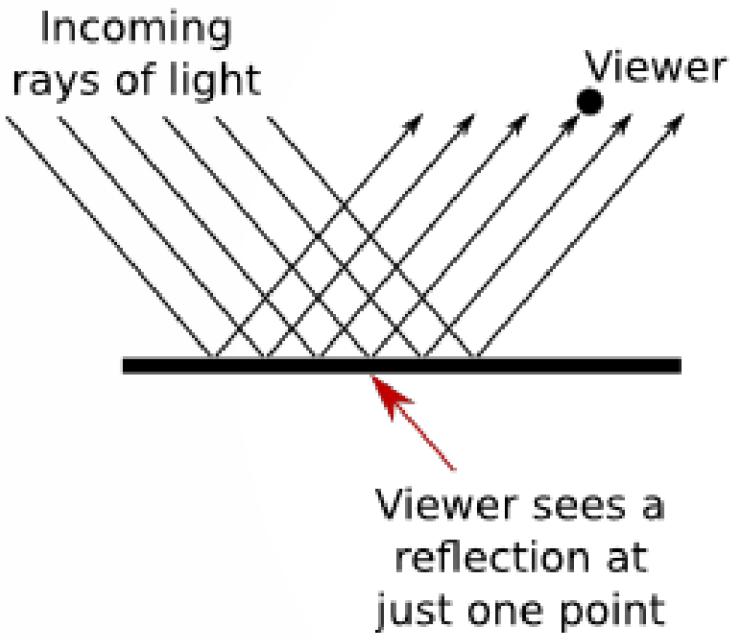
Diffuse reflection



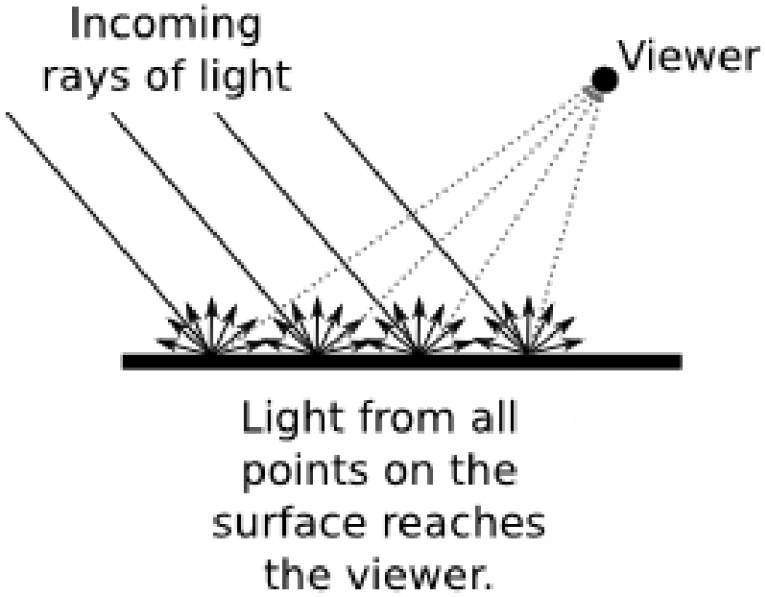
Specular reflection

Light Sources

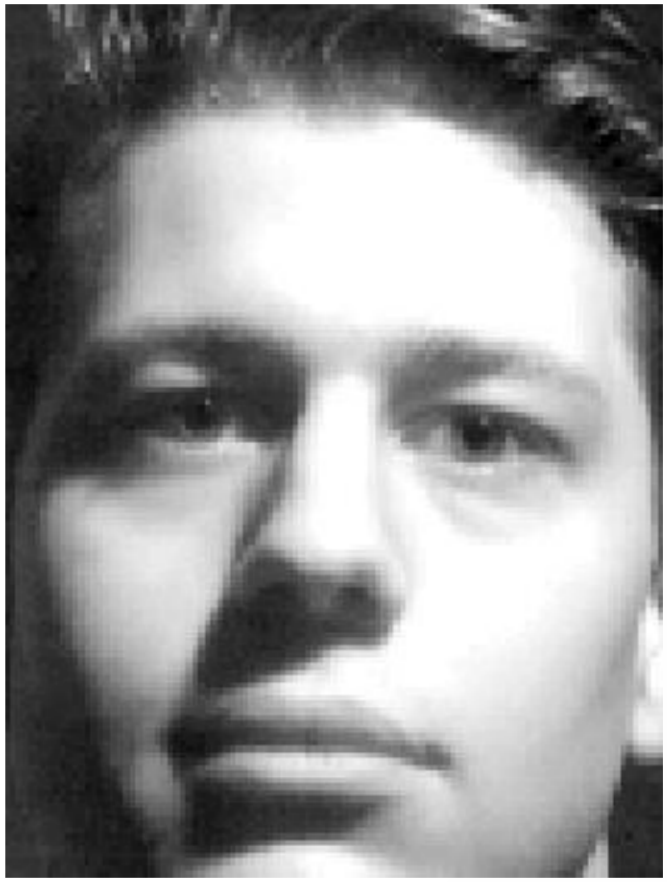
Specular Reflection



Diffuse Reflection



Lighting Effects

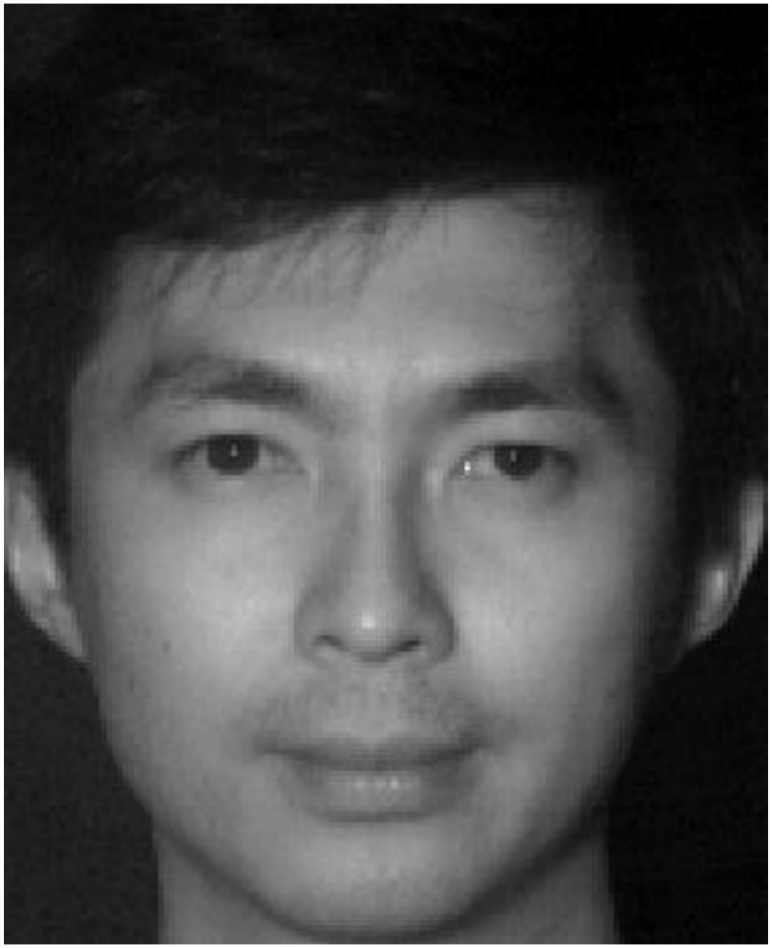


Lighting affects Appearance

Lighting Effects

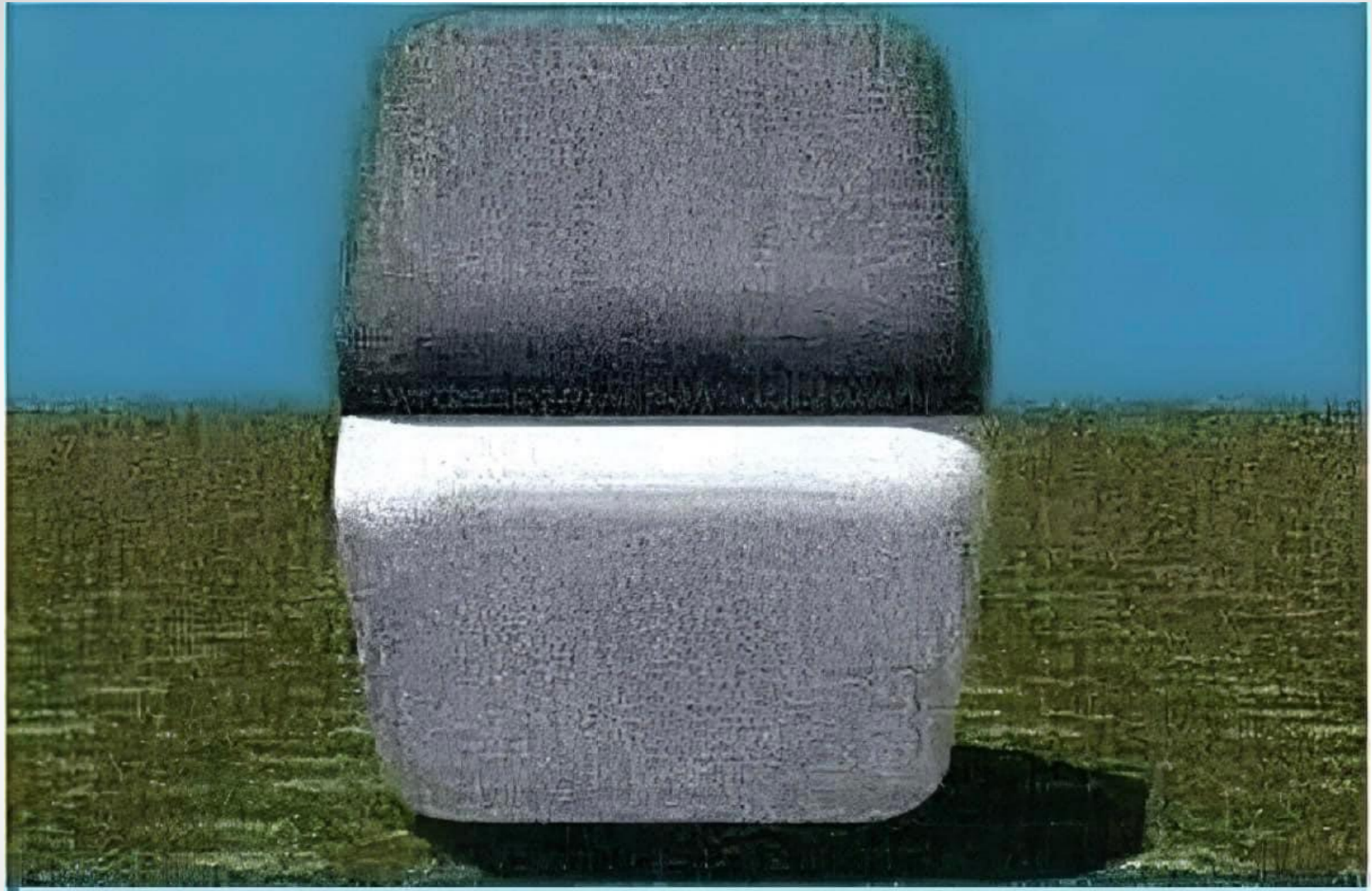
Lighting affects Appearance

Lighting Effects



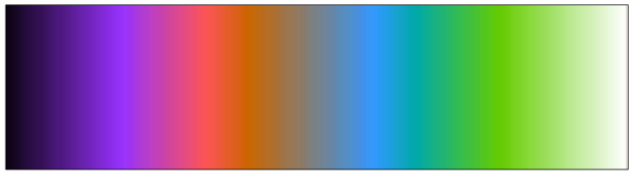
Lighting affects Appearance

Color



Both blocks are grey in color. Use your finger to cover the middle line. Light plays an important role in colors!

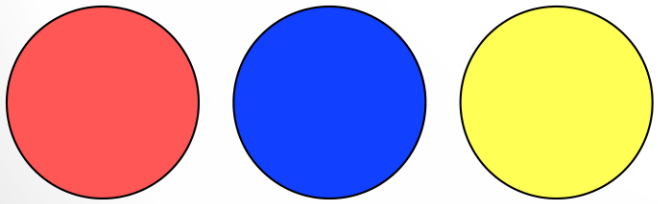
Natural Color System



146,116 individual colors

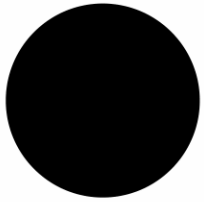
+

Presence or Absence of light

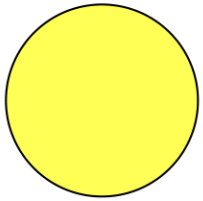


Computer Color System

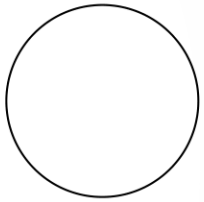
… let's try to we think this way:



BLACK
No light is reflected
All the light is absorbed



YELLOW
Yellow light is reflected
The rest of light is absorbed



WHITE
All the light is reflected
No light is absorbed

Conclusion:
If the model is based on *reflection*, then black is 0 (**RGB**).
If the model is based on *absorption*, then black is 1 (**CMY**).

Color Models

A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components.

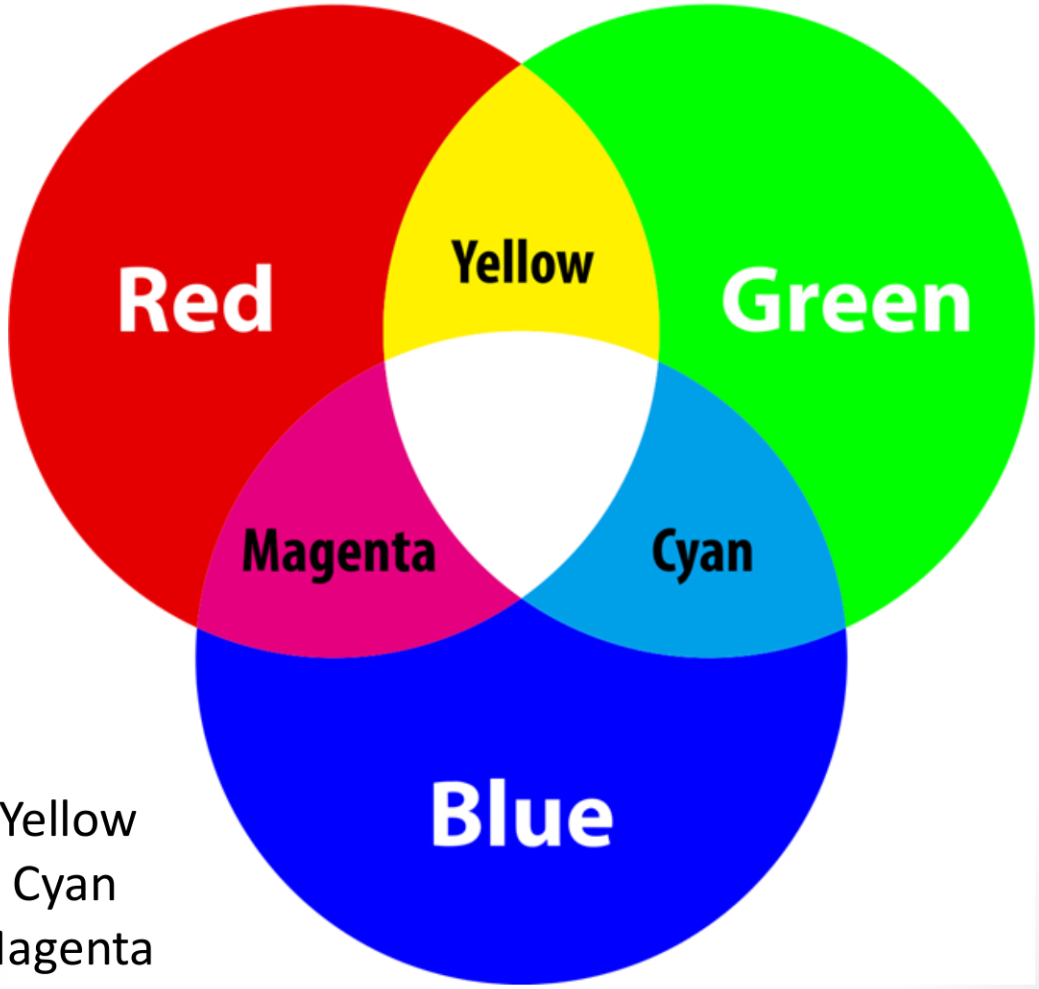
Color Models

- RGB
- CMY(K)
- HSV
- YIQ

RGB Color Model

- The RGB color model is an *additive* color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors.
- The name of the model comes from the initials of the three additive primary colors: red, green and blue.
- All other colors are produced by the proportional ratio of these three colors only.
- 0 represents the **black** and as the value increases the color intensity increases.
- Used in color CRT monitor, Digital Image Processing (DIP) , openCV and online logos.

RGB Color Model



Color combination:

Green (255) + Red (255) = Yellow

Green (255) + Blue (255) = Cyan

Red (255) + Blue (255) = Magenta

Red (255) + Green (255) + Blue (255) = White

There's No Red in this image. It's just your mind filling in the gaps.
Don't believe then zoom in to see for yourself.

RGB Color Model

RGB IMAGE

RED GREEN BLUE

RGB Color Model: Geometrical Structure

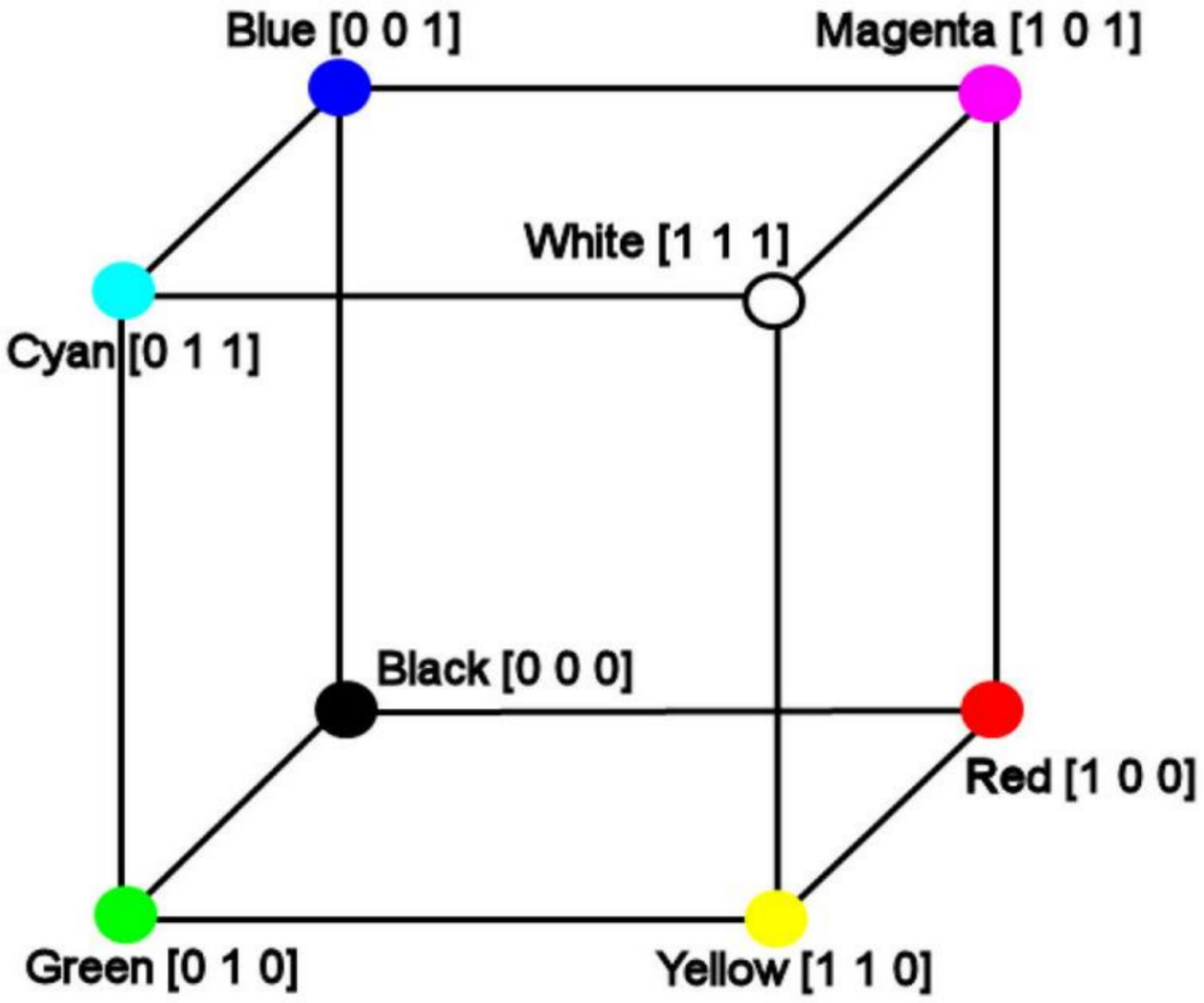


Figure: RGB Color Model Cube

RGB Color Model: Geometrical Structure

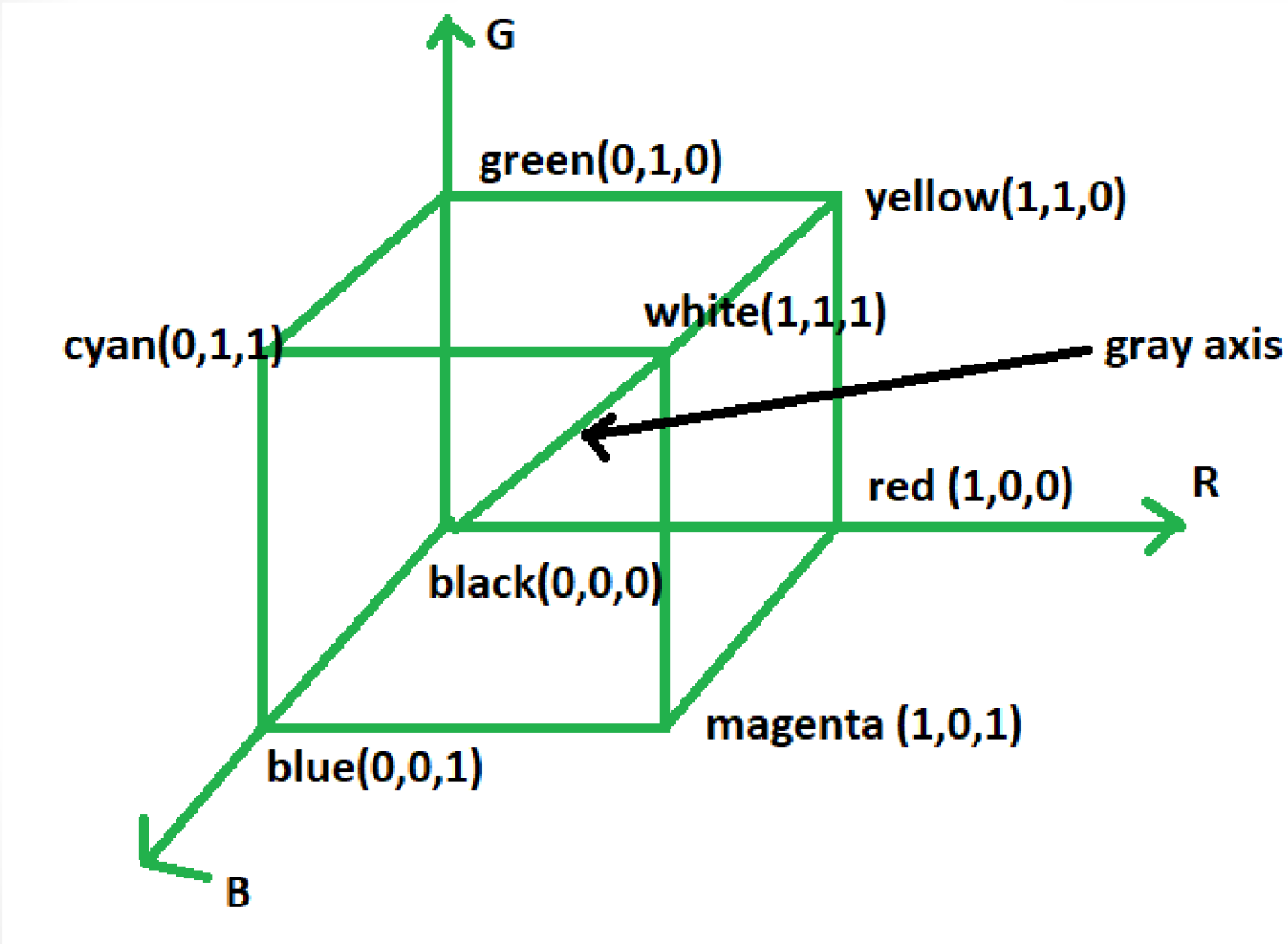


Figure: RGB Color Model Cube

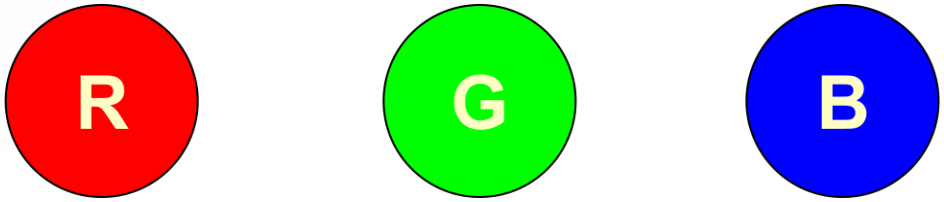
RGB Color Model: Geometrical Structure

- Each primary color (R,G,B) can take an intensity value ranging from 0 (lowest) to 1 (highest).
- Mixing these three primary colors at different intensity levels produces a variety of colors.
- The collection of all the colors obtained by such a linear combination of red, green and blue forms the ***cube*** shaped RGB color space.
- The corner of RGB color cube that is at the origin of the coordinate system corresponds to **black**, whereas the corner of the cube that is diagonally opposite to the origin represents **white**.

RGB Color Model: Geometrical Structure

- The diagonal line connecting black and white corresponds to all the gray colors between black and white, which is also known as ***gray axis***.
- In the RGB color model, an arbitrary color within the cubic color space can be specified by its color coordinates: (r, g, b).
- **Example:**
  - (0, 0, 0) for **black**, (1, 1, 1) for **white**,
  - (1, 1, 0) for **yellow**, (0.7, 0.7, 0.7) for **gray**

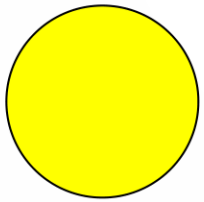
RGB Color Model



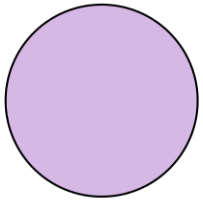
RGB (Red, Green, Blue)

Red (255,0,0)
Green (0,255,0)
Blue (0,0,255)

Each color has a range from **0 to 255**, i.e. 256 levels



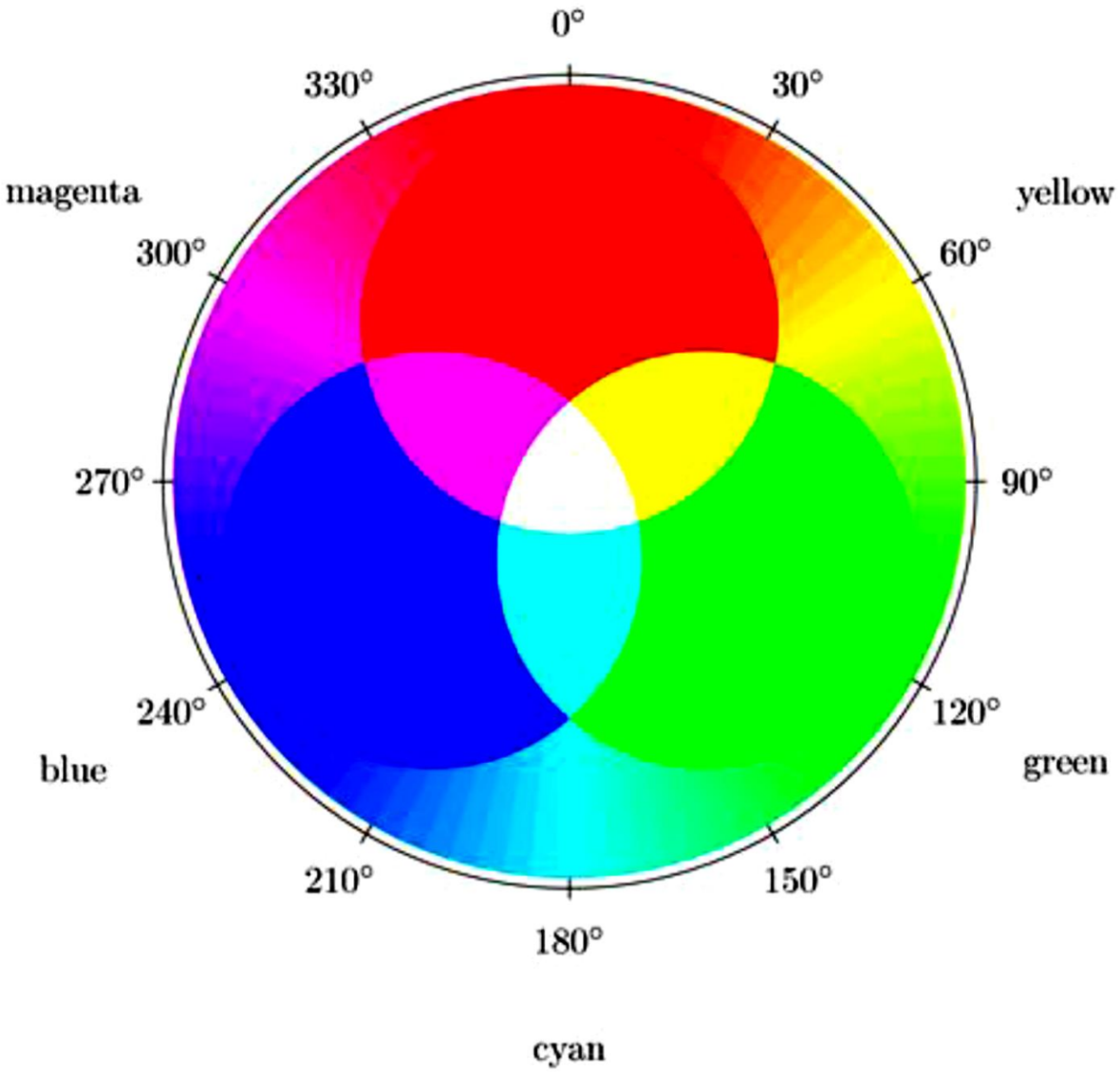
(255,255,0)



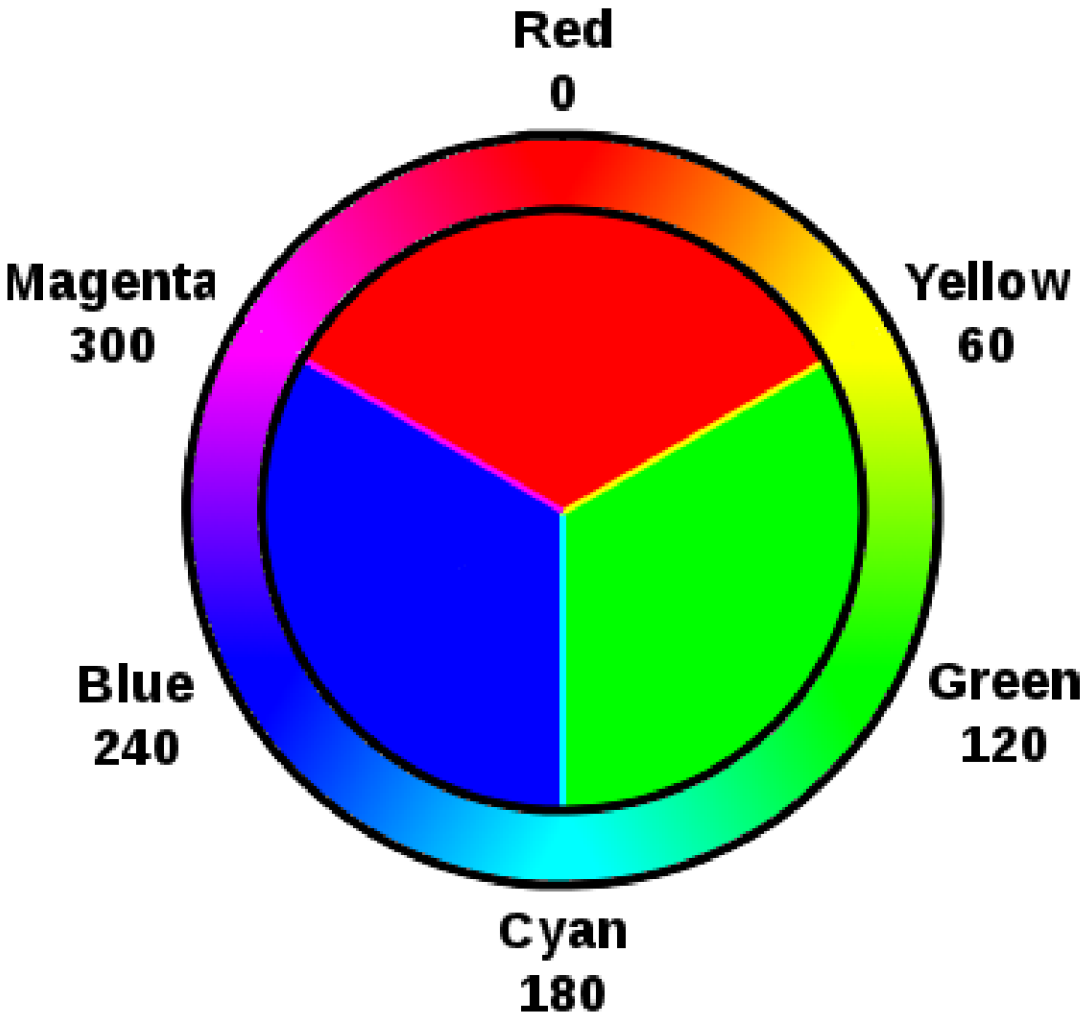
(215,185,229)

There are 256 x 256 x 256 = 16,777,216 possible colors.

RGB Color Model



RGB Color Model



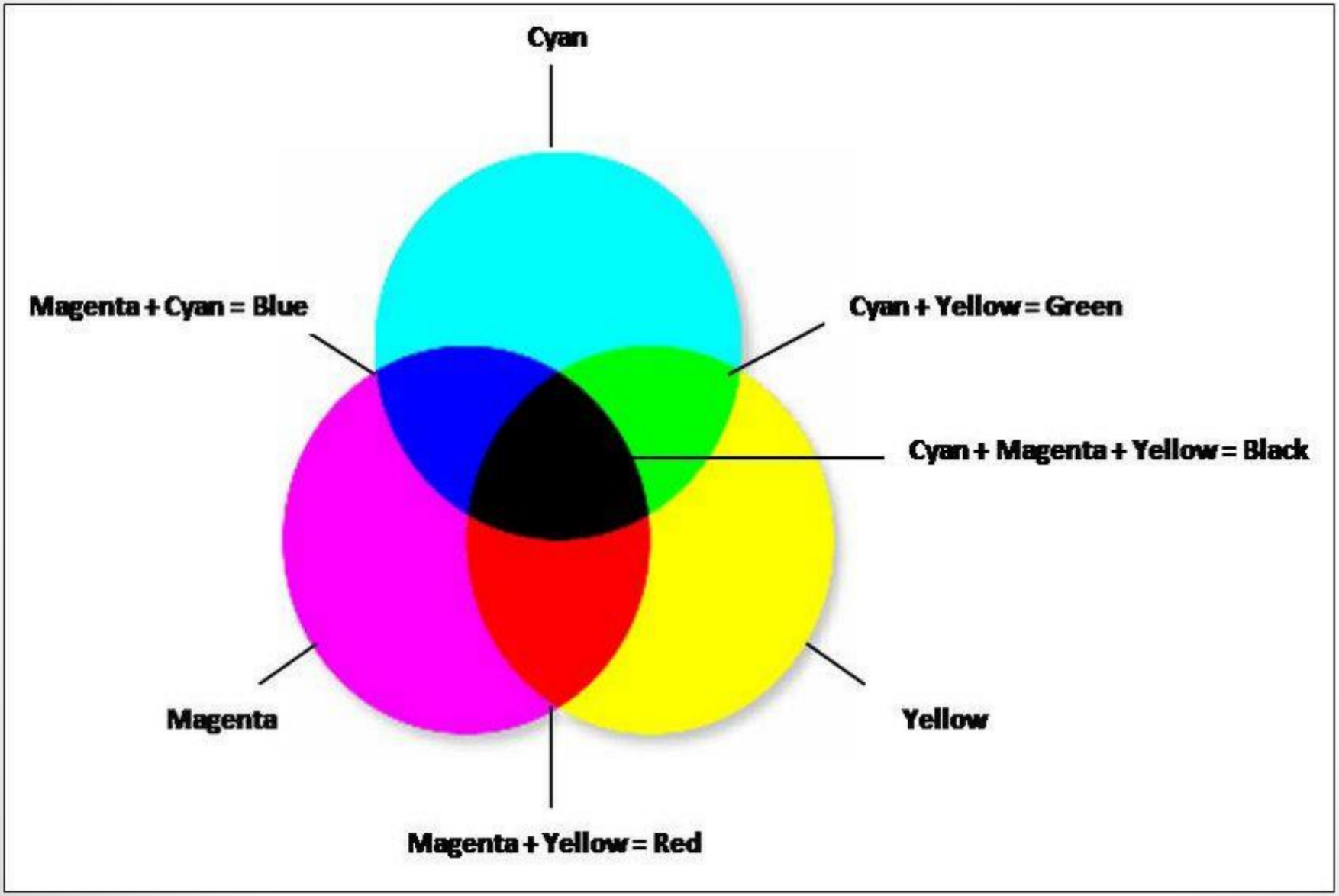
CMYK Color Model

- CMYK color model is widely used in printers.
- It stands for Cyan, Magenta, Yellow and Black (Key).
- It is a *subtractive* color model.
- 0 represents the primary color and 1 represents the lightest color.
- In this model, point (1, 1, 1) represents black, and (0,0,0) represents white.
- It is a subtractive model thus the value is subtracted from 1 to vary from least intense to a most intense color value.

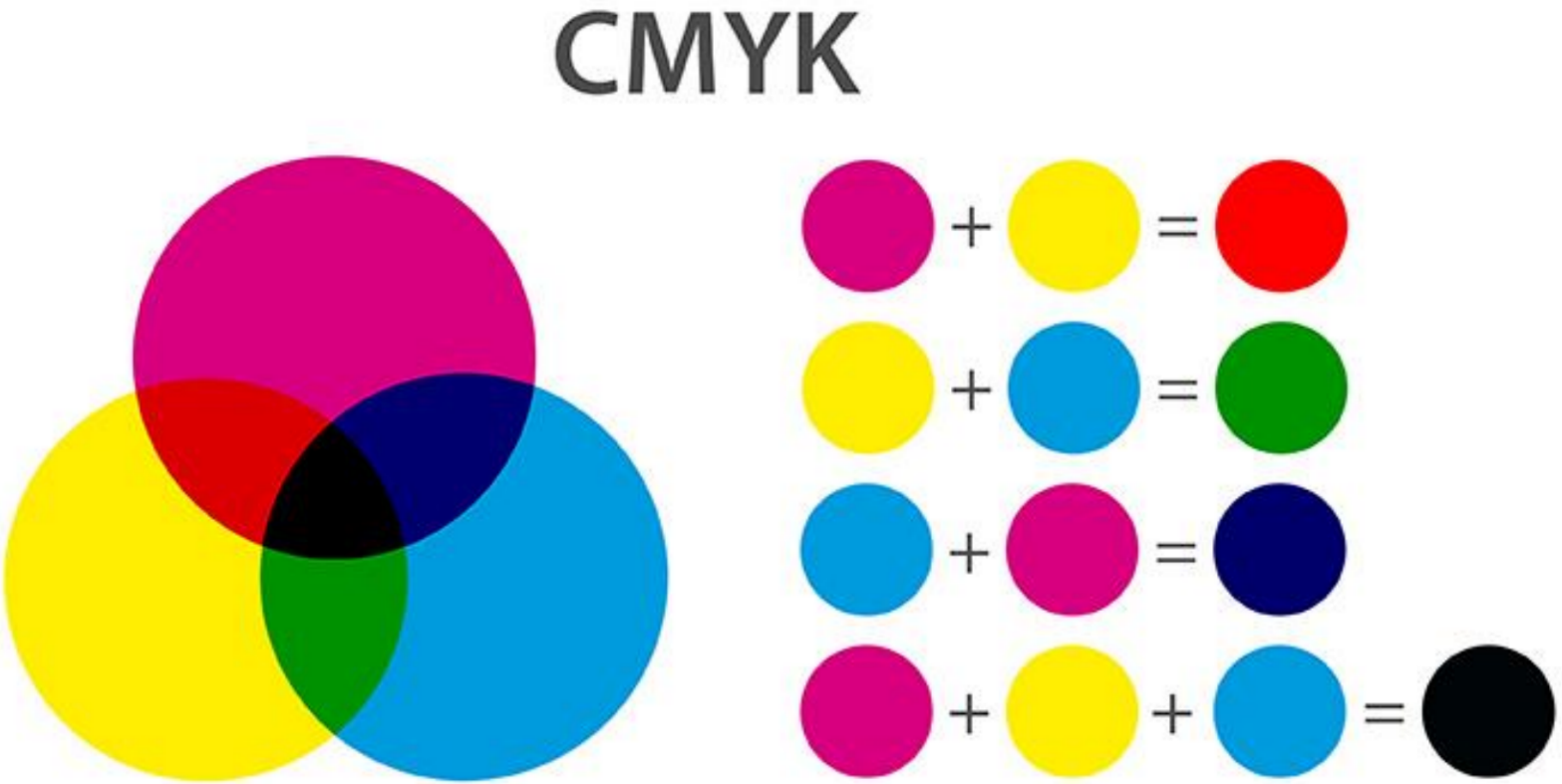
1 - RGB = CMY

- **Cyan** is negative of Red.
- **Magenta** is negative of Green.
- **Yellow** is negative of Blue.

CMYK Color Model



CMYK Color Model

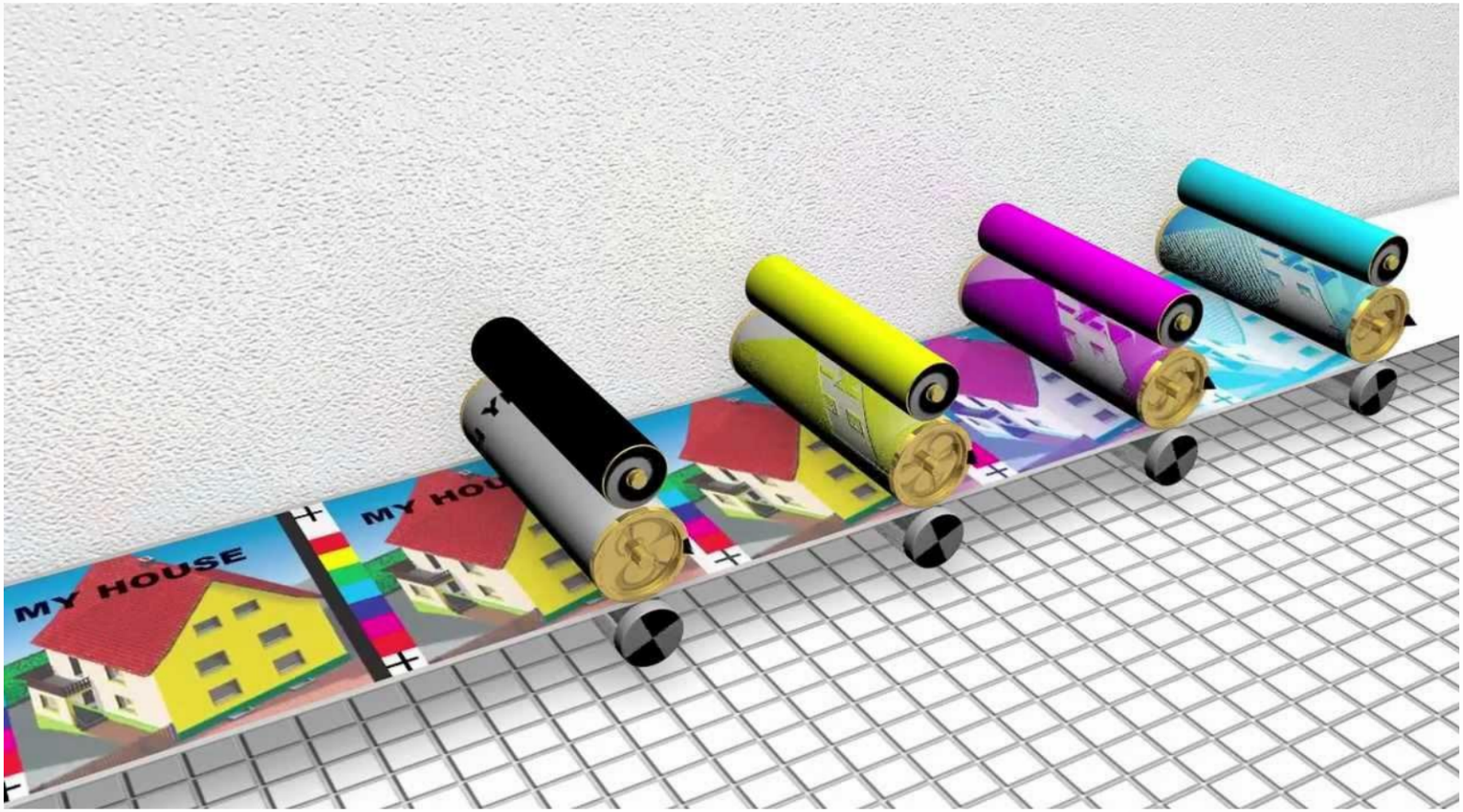


CMYK Color Model

CMYK IMAGE

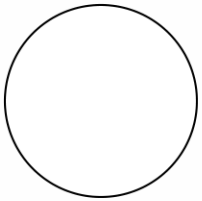
CYAN MAGENTA YELLOW BLACK

CMYK Color Model

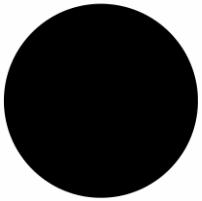


How Colors Combine ?

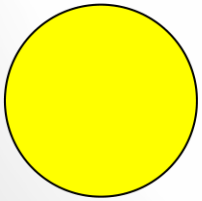
- The colors may also be written as range of:
  - binary numbers from 00000000 to 11111111
  - hexadecimal from 00 to FF



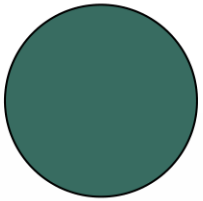
(255,255,255)
(#FFFFFF)



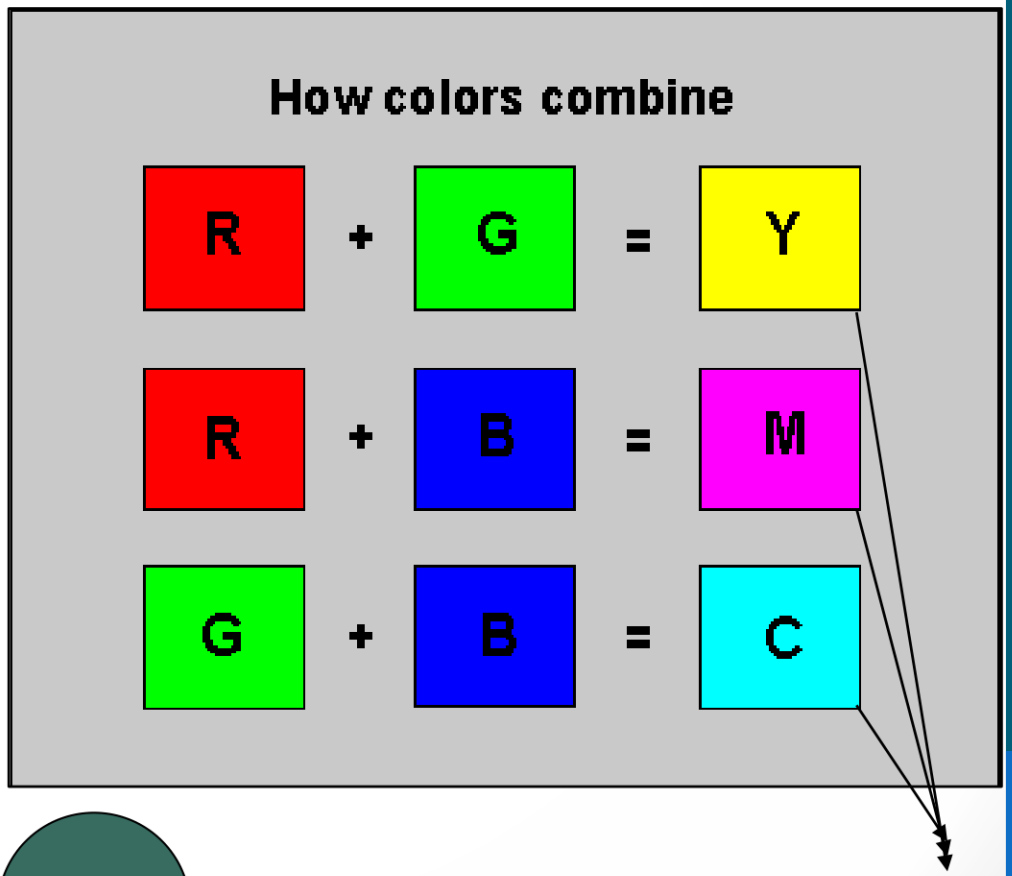
(0,0,0)
(#000000)



(255,255,0)
(#FFFF00)



(56,108,98)
(#386C62)



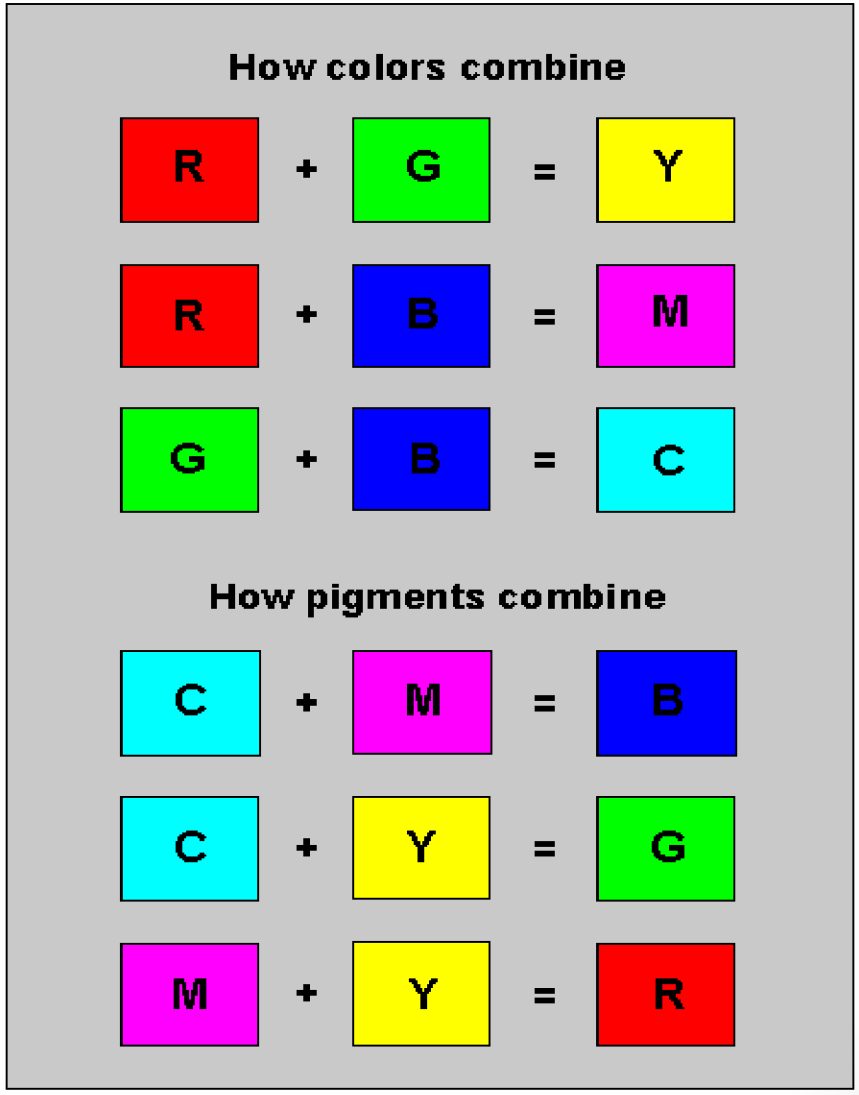
How Colors and Pigments Combine ?

RGB describes colours:

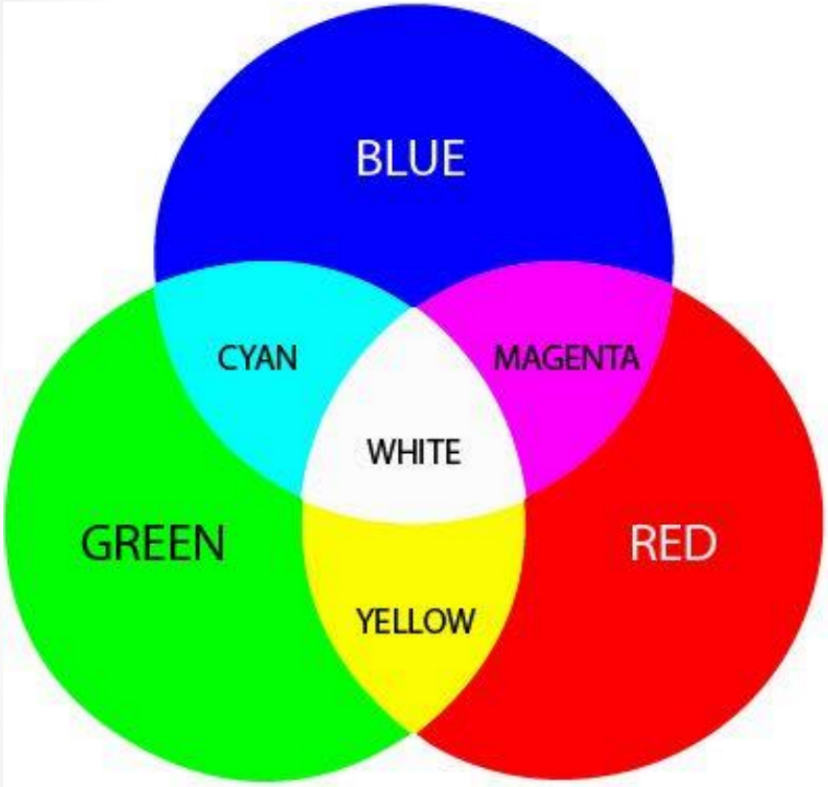
- R – Red
- G – Green
- B – Blue

CMYK describes pigments:

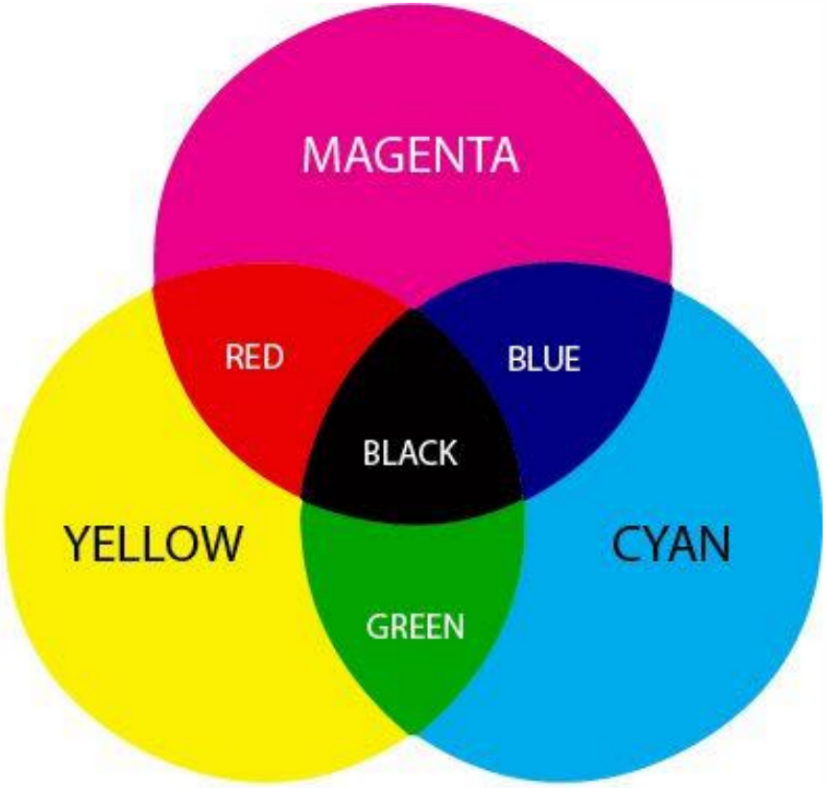
- C – Cyan (Aqua)
- M – Magenta (Pink)
- Y – Yellow
- K – Black (Key Color)



Additive vs. Subtractive Color Model



RGB
Additive (Light)



CMYK
Subractive (Ink)

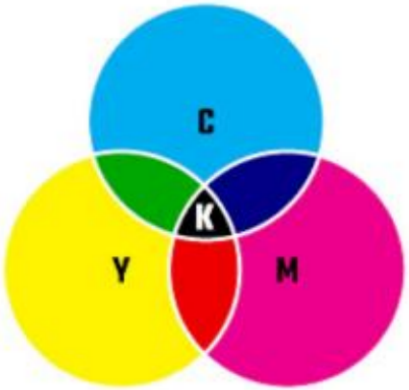
Additive vs. Subtractive Color Model

ADDITIVE COLORS

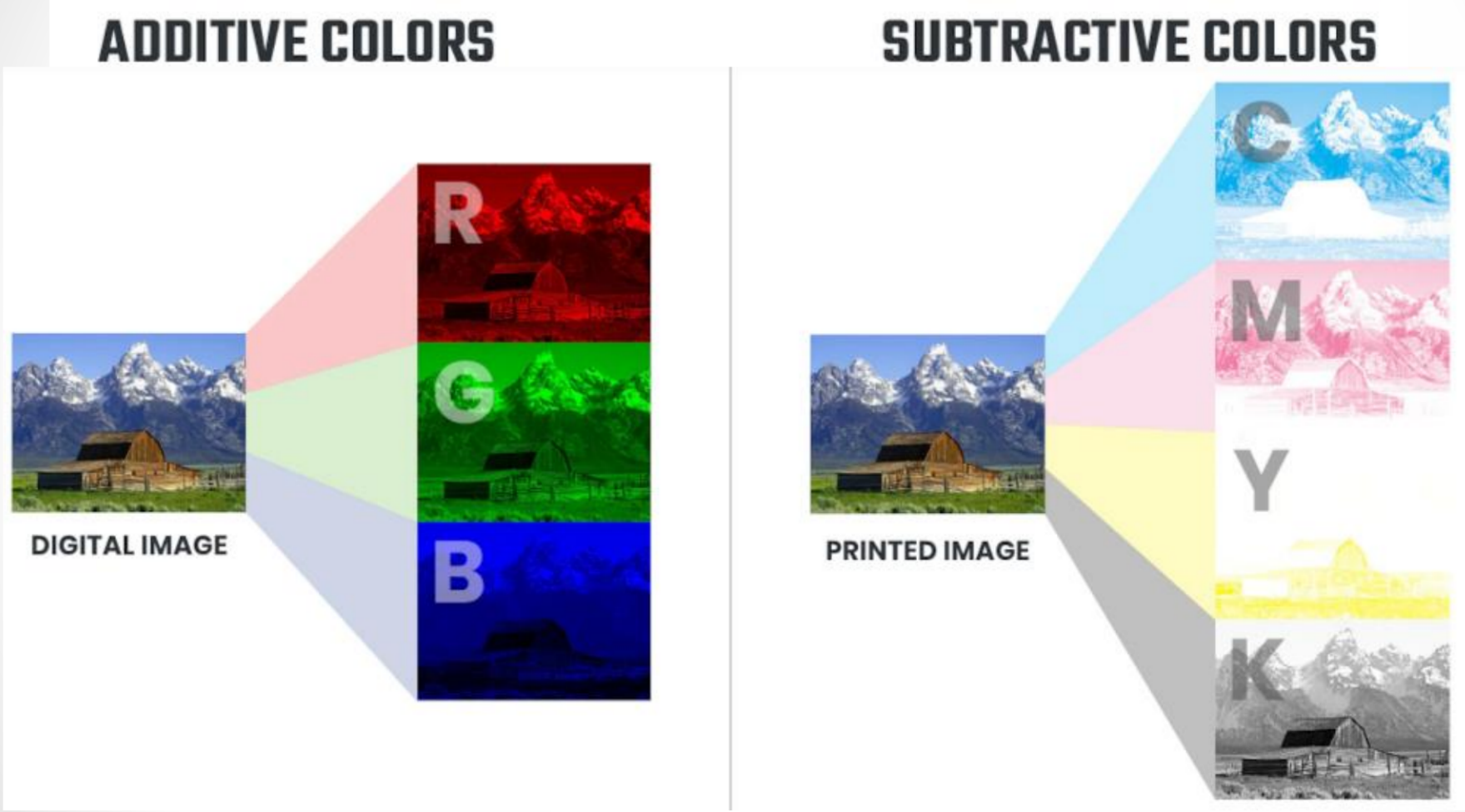
Uses light to display color by adding red, green and blue (RGB)

SUBTRACTIVE COLORS

Uses ink to display color by mixing cyan, magenta, yellow and black (CMYK).



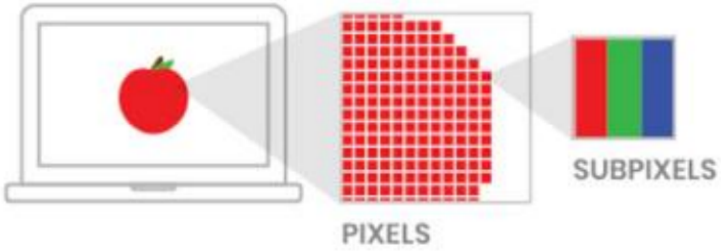
Additive vs. Subtractive Color Model



Additive vs. Subtractive Color Model

ADDITIVE COLORS

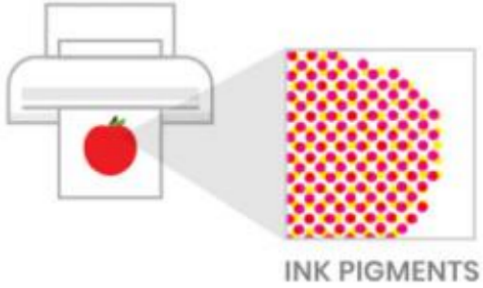
USE RGB FOR DIGITAL



e.g., web graphics or videos

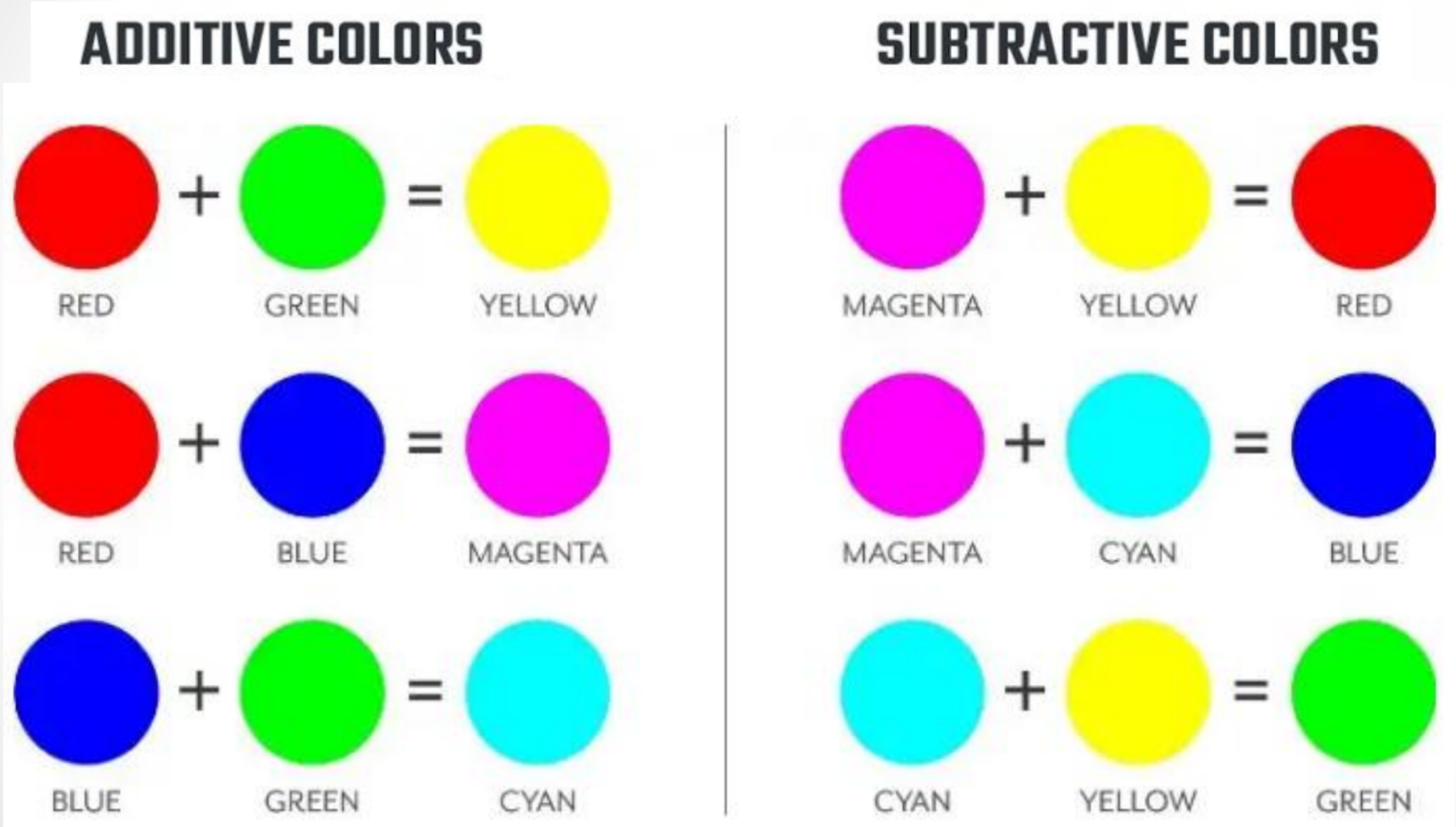
SUBTRACTIVE COLORS

USE CMYK FOR PRINT

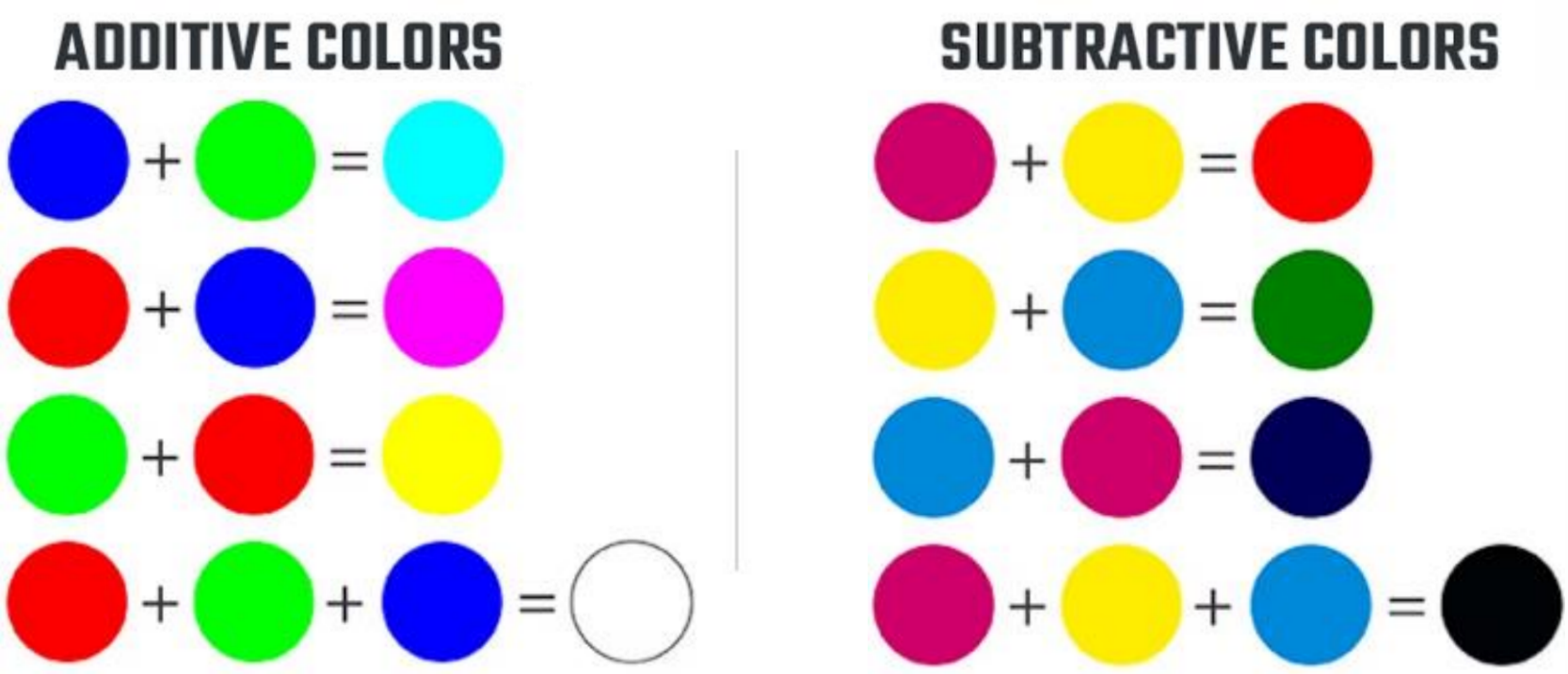


e.g., posters or newsletters

Additive vs. Subtractive Color Model



Additive vs. Subtractive Color Model



Additive vs. Subtractive Color Model

Parameters	Additive colors	Subtractive colors
Definition	Additive colors are created by mixing lights of different wavelengths. The colors act as irritants to the lenses.	Subtractive colors are formed by removing different wavelength colors.
Color Purity	Additive colors are colors which are "pure", i.e. colors add up to form white light. A RED light looks RED because it emits RED light.	Subtractive colors are "impure". You perceive RED pigment to be RED because it reflects RED light and absorbs everything except RED light falling on it.
Primary colors	Red, blue, and green	Cyan, magenta, and yellow
Transparency	Appears opaque to the eyes	Appears transparent to the eyes.

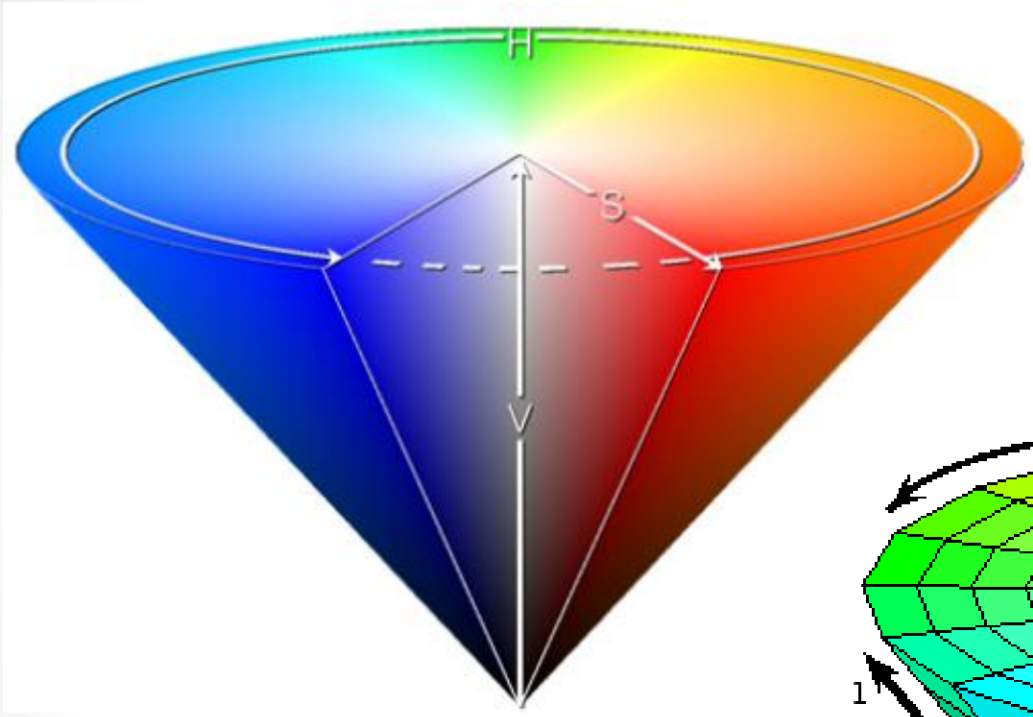
Additive vs. Subtractive Color Model

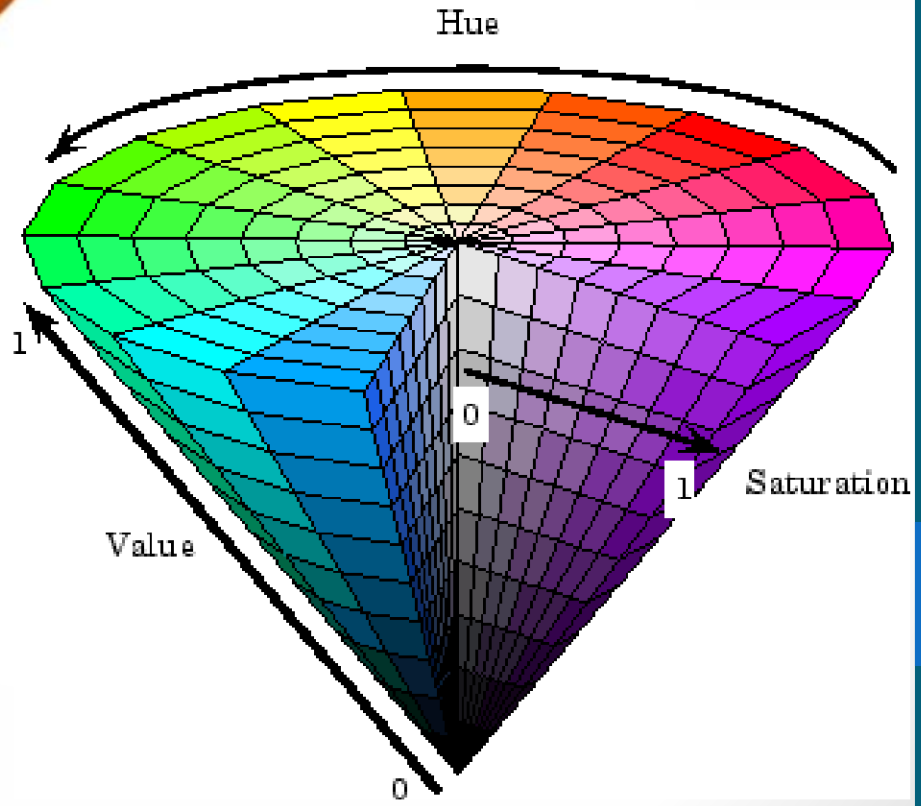
Parameters	Additive colors	Subtractive colors
Mixing of colors	Red + Green + Blue = White	Cyan + Magenta + Yellow = Black
Displaying colors	Uses light to display color	Uses ink to display color
Colors resulted from	Colors result from transmitted light	Colors result from reflected light
System involved	RGB	CMYK
Used for	For computer displays	For printed material

HSV Color Model

- This color model is based on polar coordinates, not Cartesian coordinates.
- HSV is a non-linearly transformed (skewed) version of RGB cube
- Hue, Saturation and Value (Brightness) are three channels.
- This color model does not use primary colors directly.
- It uses color in the way humans perceive them.
- HSV color models are particularly useful for selecting precise colors for art, color swatches, and digital graphics.
- HSV color is represented by a cone.

HSV Color Model





HSV Color Model

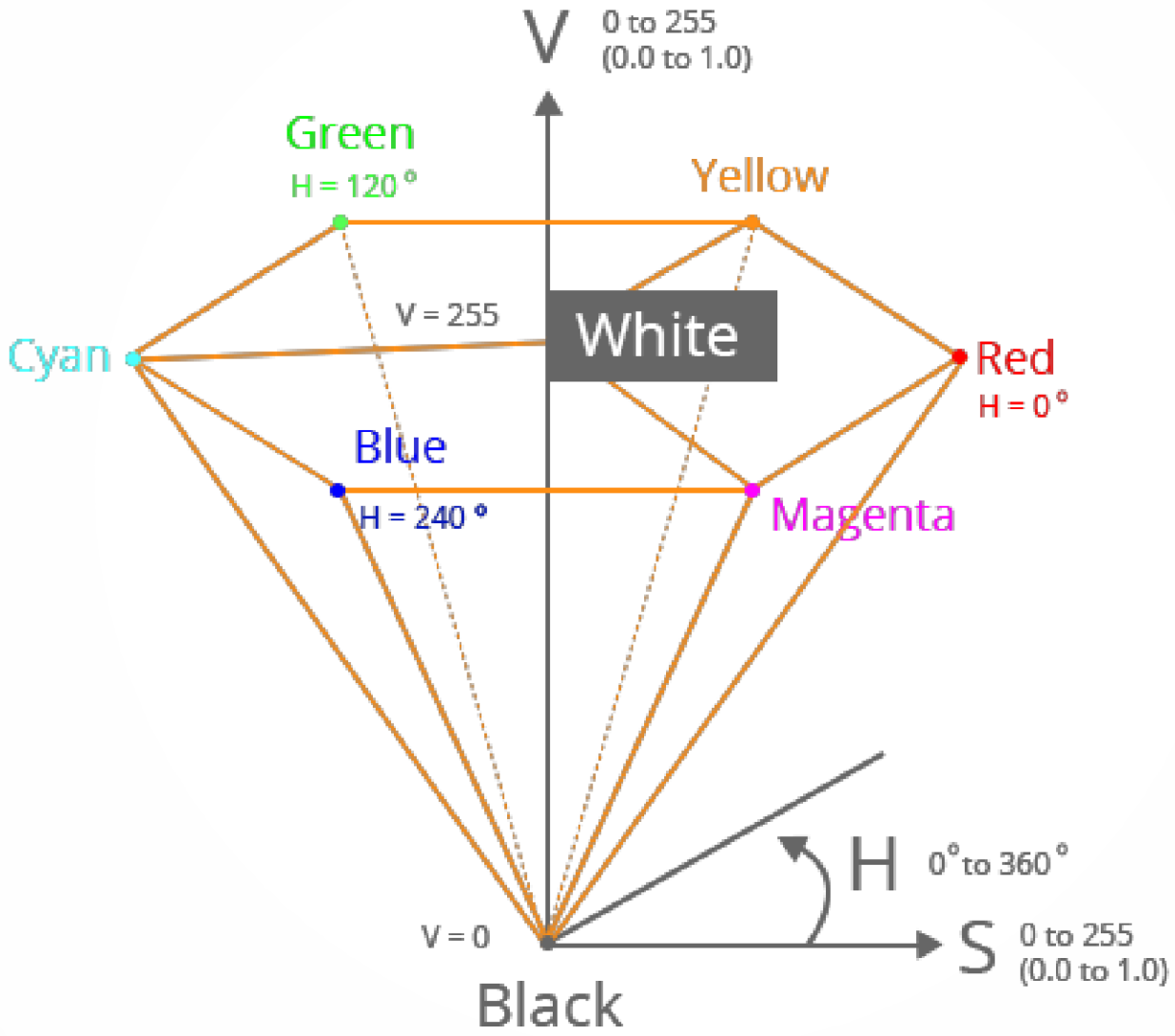


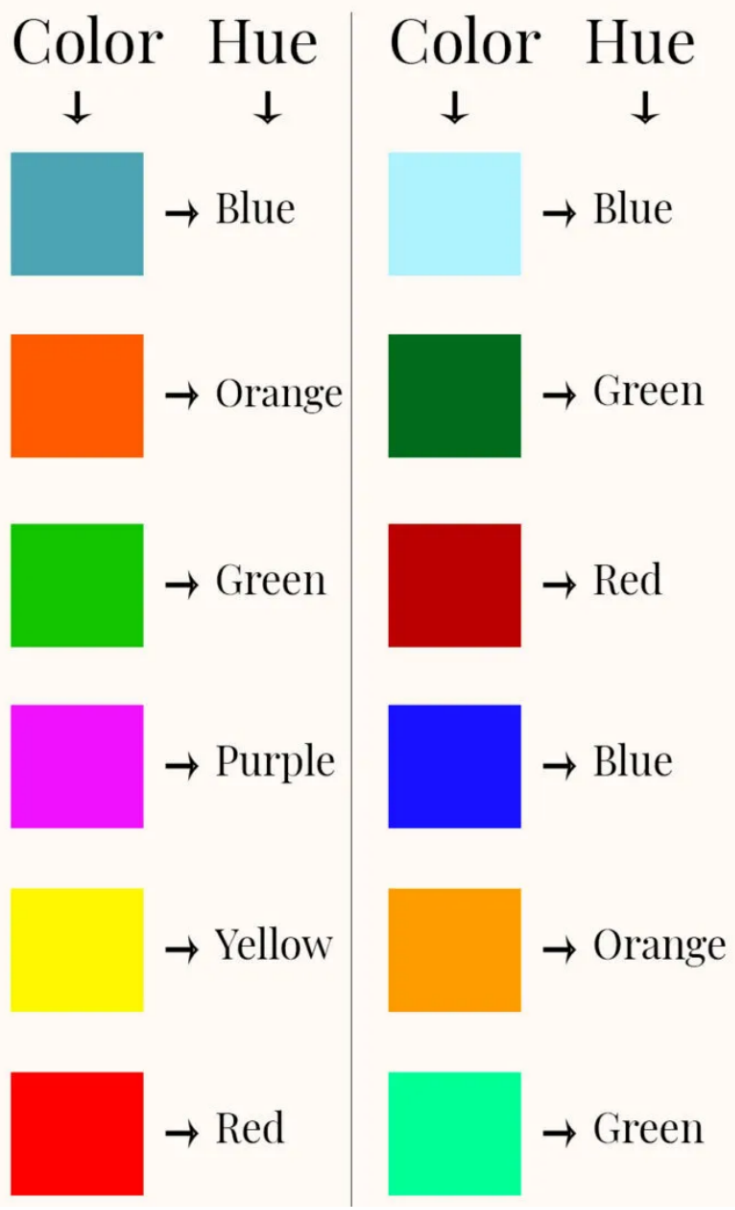
Figure: HSV color model represents each color in a six sided pyramid form

HSV Color Model

- **Hue (Color/Tint)**

  - Hue is a color component.
  - Hue represents different colors in different angle ranges.
  - Hue is measured in degrees from 0 to 360.
    - Red falls between 0 and 60 degrees.
    - Yellow falls between 61 and 120 degrees.
    - Green falls between 121 and 180 degrees.
    - Cyan falls between 181 and 240 degrees.
    - Blue falls between 241 and 300 degrees.
    - Magenta falls between 301 and 360 degrees.

HSV Color Model



HSV Color Model

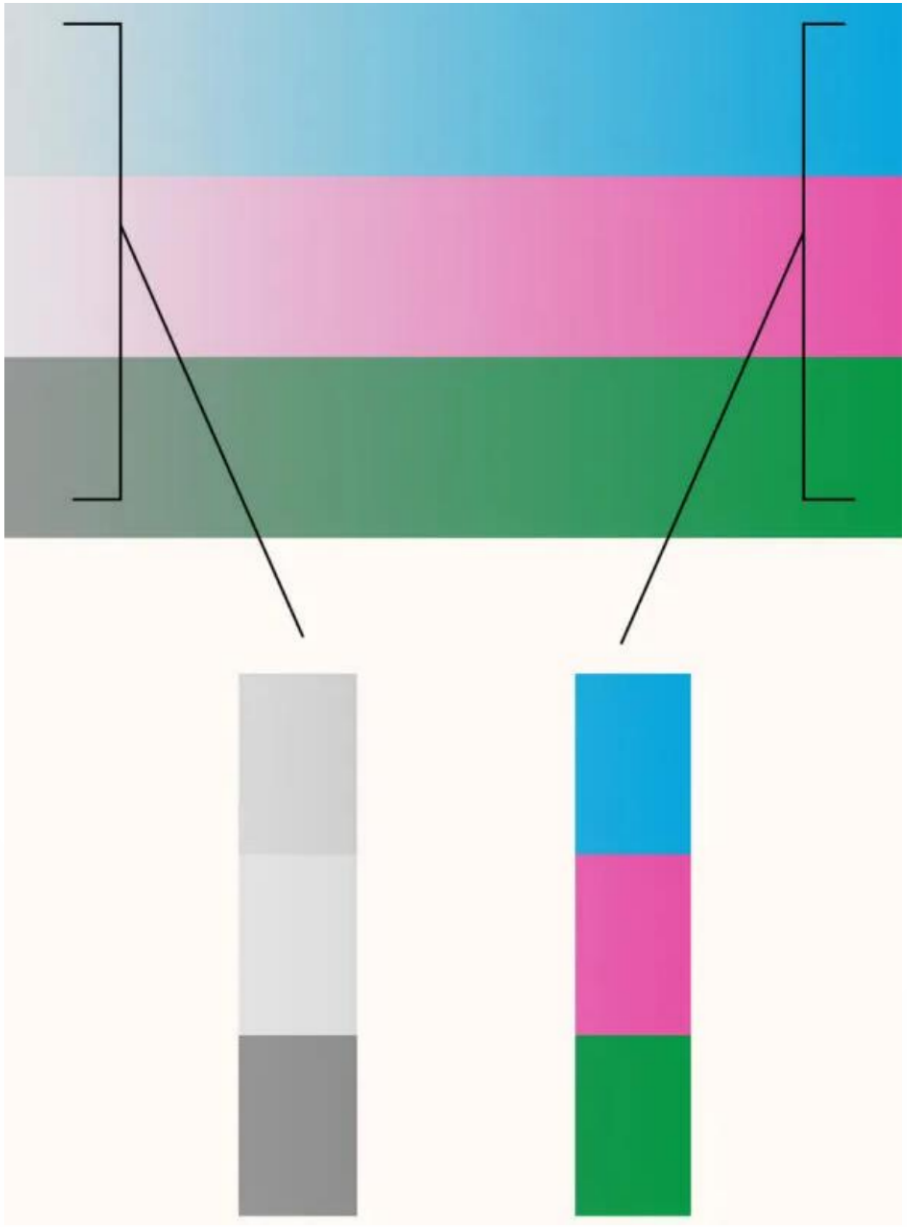
Range of Hues

HSV Color Model

Saturation (Shade/Amount of Gray)

- Saturation represents the vibrancy of the color/vividness of color
- It describes the percentage of the color.
- Saturation describes the amount of gray in a particular color, from 0 to 100 percent.
- If something is fully saturated it means that it's 100 percent that color.
- If something is completely desaturated, the color is completely grey.
- Reducing this component toward zero introduces more gray and produces a faded effect.
- Saturation appears as a range from 0 to 1, where 0 is gray, and 1 is a primary color.

HSV Color Model



HSV Color Model

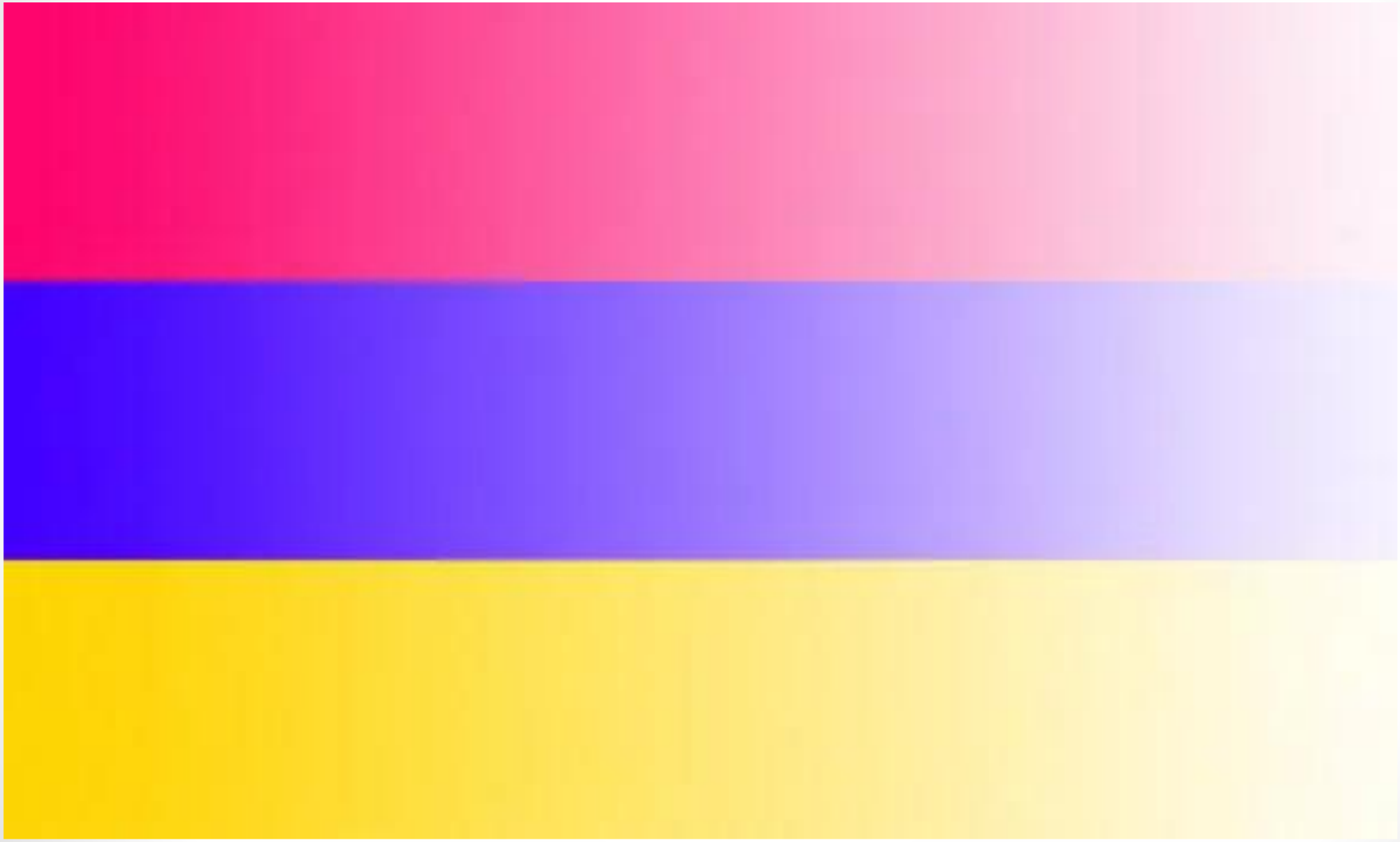
- **Value** (**Luminance/Brightness**)

  - The value represents the intensity of the color chosen.
  - Value works in conjunction with saturation and describes the brightness or intensity of the color
  - Its value lies in percentage from 0 to 100.
  - 0 is completely black and 100 is the brightest and reveals the color.

HSV Color Model

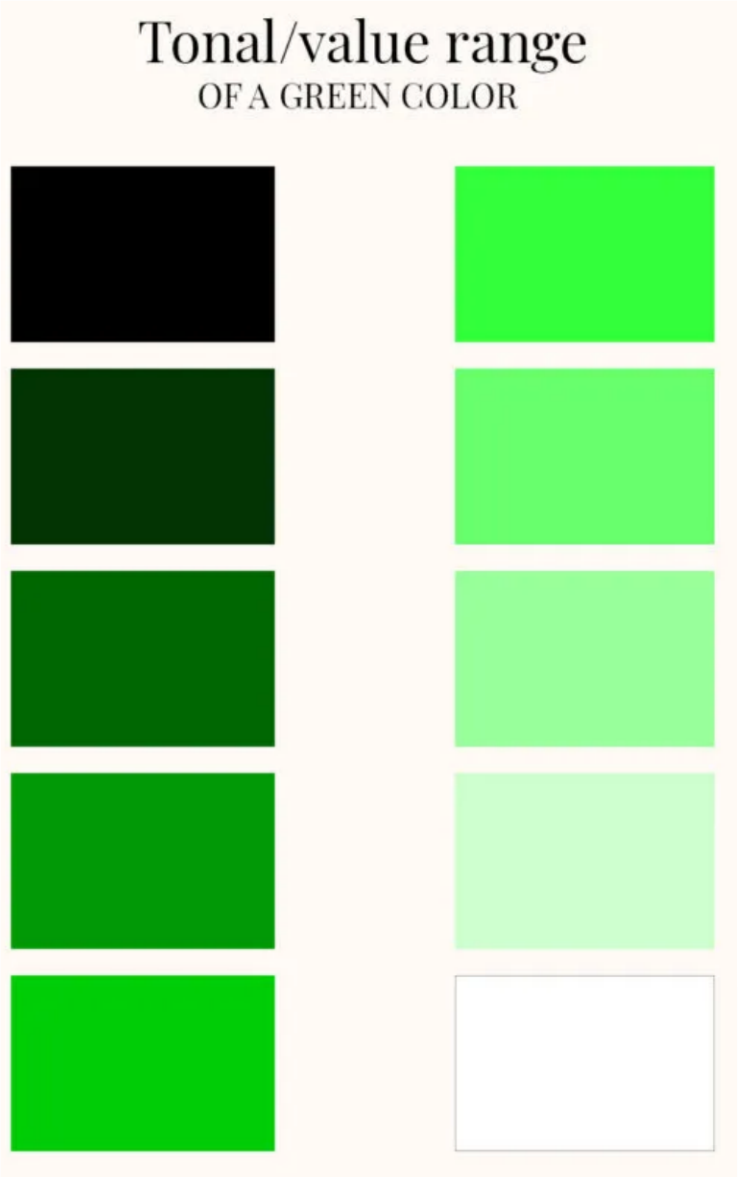
Value (0% brightness)

HSV Color Model



Value (100% brightness)

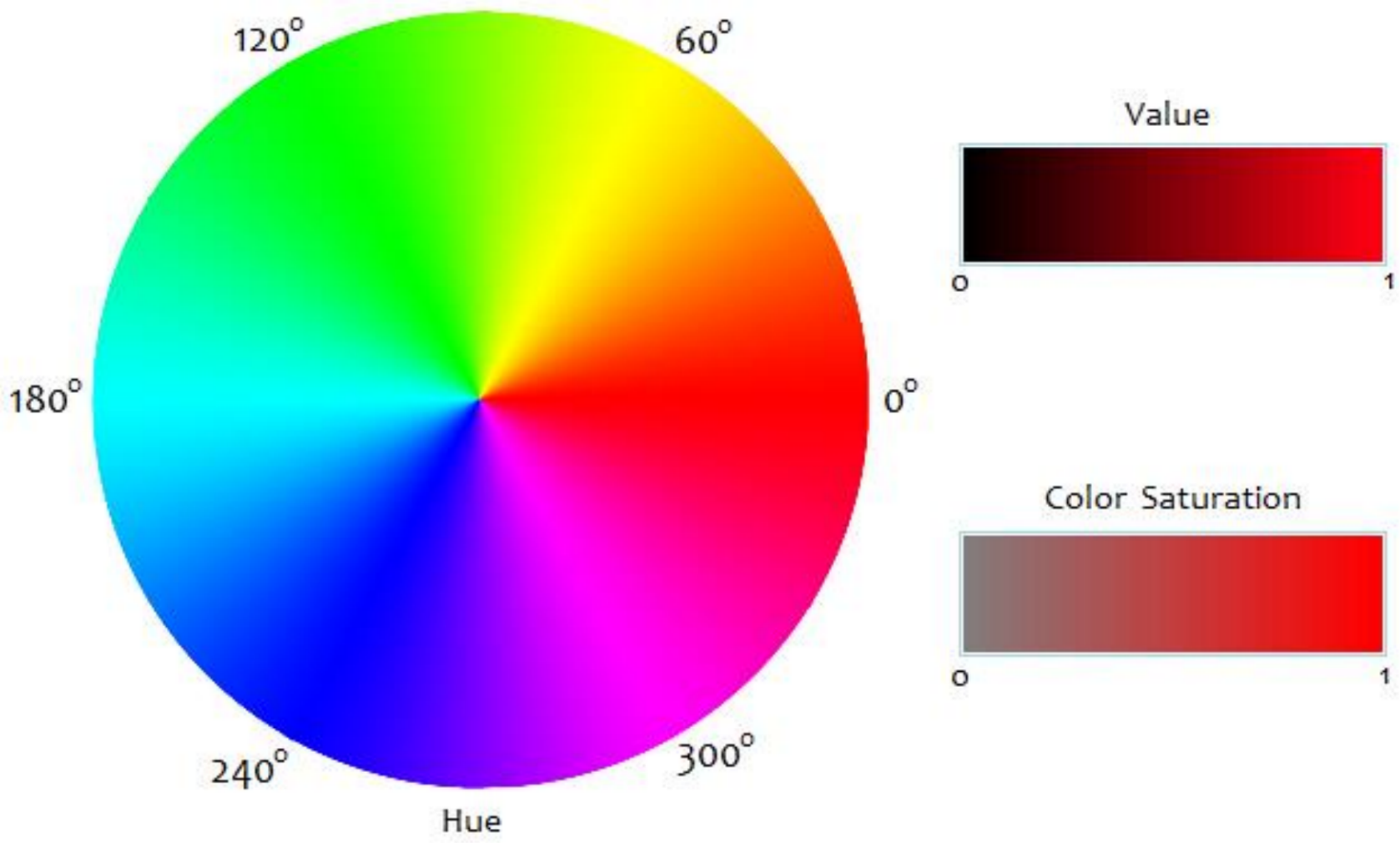
HSV Color Model



HSV Color Model

Any color (hue) + white are also known as tints.
Any color (hue) + black are also known as shades.

Conversion between RGB and HSV Color Models



Converting RGB to HSV

Given three numbers R, G, and B (each between 0 and 255), you can first define m and M with the relations

$M = \max\{R, G, B\}$
$m = \min\{R, G, B\}.$

And then V and S are defined by the equations

$V = M/255$
$S = 1 - m/M$ if $M > 0$
$S = 0$ if $M = 0.$

The hue H is defined by the equations

$H = \cos^{-1}[\ (R - \frac{1}{2}G - \frac{1}{2}B)/\sqrt{(R^2 + G^2 + B^2 - RG - RB - GB)}\ ]$ if **$G \geq B$**, or
$H = 360 - \cos^{-1}[\ (R - \frac{1}{2}G - \frac{1}{2}B)/\sqrt{(R^2 + G^2 + B^2 - RG - RB - GB)}\ ]$ if **$B > G$**.

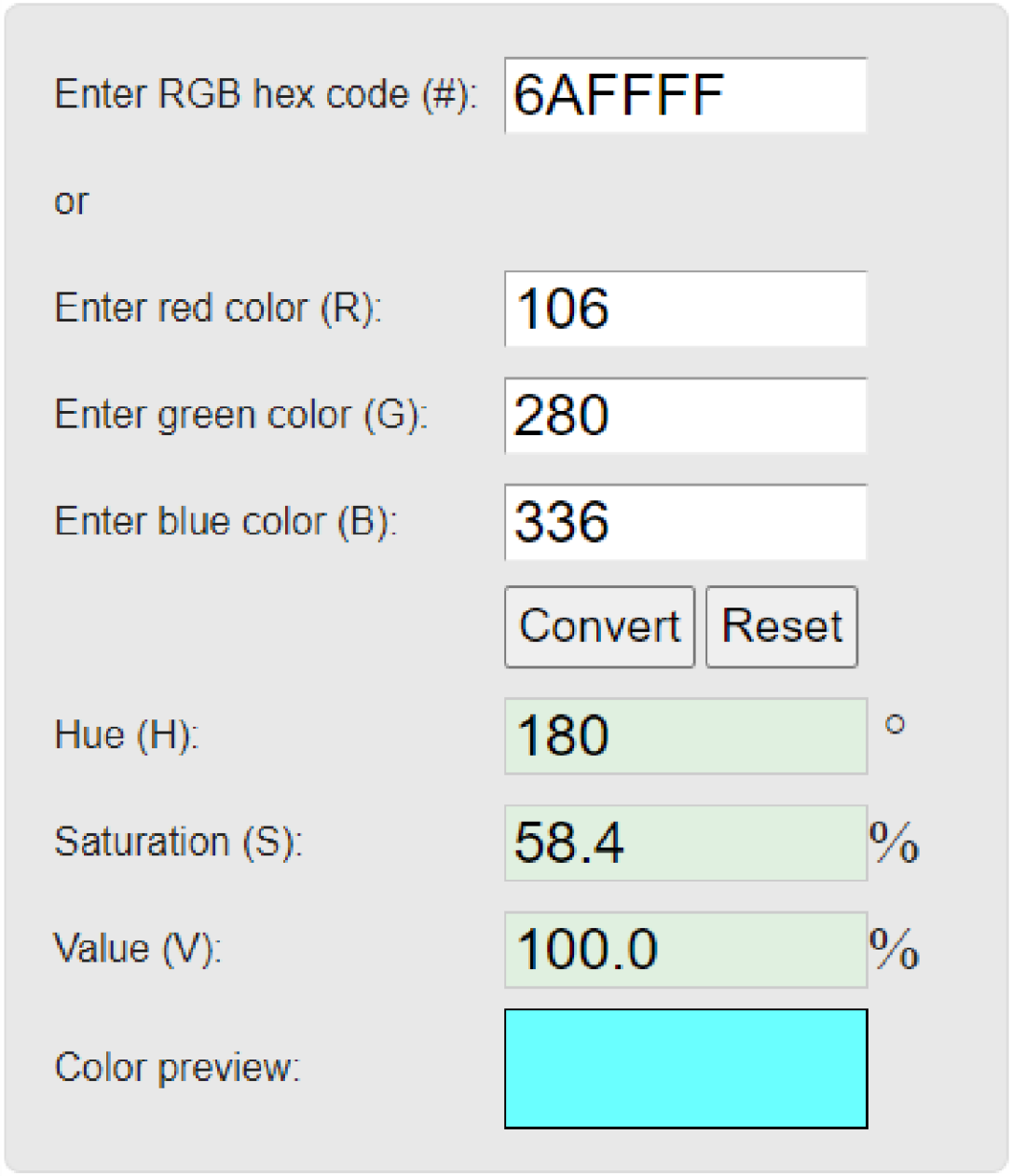
Inverse cosine is calculated in degrees.

Converting RGB to HSV

RGB to HSV color table

Color	Color name	Hex	(R,G,B)	(H,S,V)
	Black	#000000	(0,0,0)	(0°,0%,0%)
	White	#FFFFFF	(255,255,255)	(0°,0%,100%)
	Red	#FF0000	(255,0,0)	(0°,100%,100%)
	Lime	#00FF00	(0,255,0)	(120°,100%,100%)
	Blue	#0000FF	(0,0,255)	(240°,100%,100%)
	Yellow	#FFFF00	(255,255,0)	(60°,100%,100%)
	Cyan	#00FFFF	(0,255,255)	(180°,100%,100%)
	Magenta	#FF00FF	(255,0,255)	(300°,100%,100%)
	Silver	#BFBFBF	(191,191,191)	(0°,0%,75%)
	Gray	#808080	(128,128,128)	(0°,0%,50%)
	Maroon	#800000	(128,0,0)	(0°,100%,50%)
	Olive	#808000	(128,128,0)	(60°,100%,50%)
	Green	#008000	(0,128,0)	(120°,100%,50%)
	Purple	#800080	(128,0,128)	(300°,100%,50%)
	Teal	#008080	(0,128,128)	(180°,100%,50%)
	Navy	#000080	(0,0,128)	(240°,100%,50%)

Converting RGB to HSV



Converting HSV to RGB

Given the values of H, S, and V, you can first compute m and M with the equations

$M = 255V$
$m = M(1-S)$.

Now compute another number, z, defined by the equation
$z = (M-m)[1 - |(H/60)\text{mod}\_2 - 1|]$,

where mod_2 means division modulo 2. For example, if H = 135, then $(H/60)\text{mod}\_2 = (2.25)\text{mod}\_2 = 0.25$. In modulo 2 division, the output is the remainder of the quantity when you divide it by 2.

Now you can compute R, G, and B according to the angle measure of H.

There are six cases.

Converting HSV to RGB

When $0 \leq H < 60$,

$R = M$
$G = z + m$
$B = m.$

When $60 \leq H < 120$,

$R = z + m$
$G = M$
$B = m.$

When $120 \leq H < 180$,

$R = m$
$G = M$
$B = z + m.$

When $180 \leq H < 240$,

$R = m$
$G = z + m$
$B = M.$

When $240 \leq H < 300$,

$R = z + m$
$G = m$
$B = M.$

And if $300 \leq H < 360$,

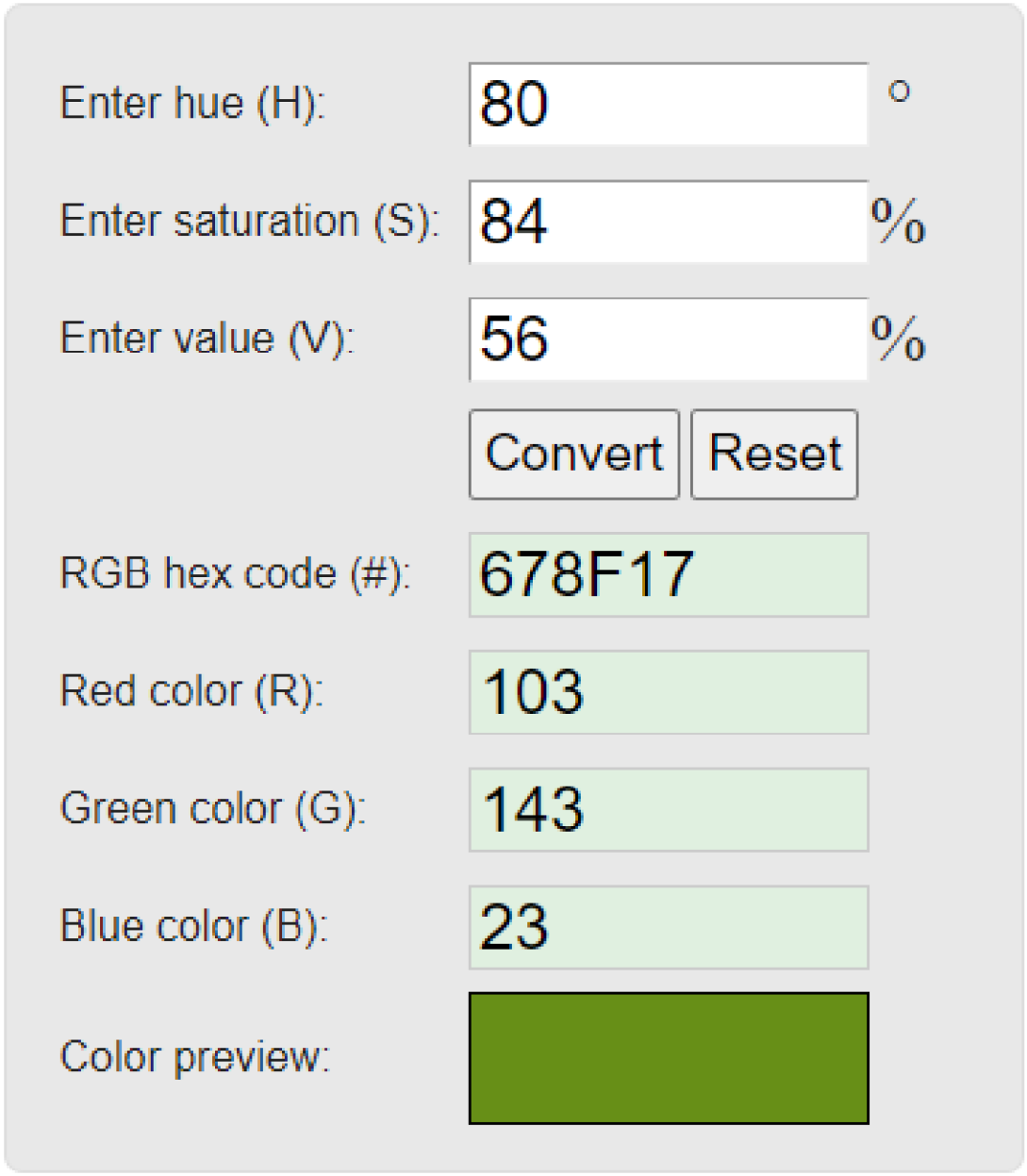
$R = M$
$G = m$
$B = z + m.$

Converting HSV to RGB

HSV to RGB color table

Color	Color name	(H,S,V)	Hex	(R,G,B)
	Black	(0°,0%,0%)	#000000	(0,0,0)
	White	(0°,0%,100%)	#FFFFFF	(255,255,255)
	Red	(0°,100%,100%)	#FF0000	(255,0,0)
	Lime	(120°,100%,100%)	#00FF00	(0,255,0)
	Blue	(240°,100%,100%)	#0000FF	(0,0,255)
	Yellow	(60°,100%,100%)	#FFFF00	(255,255,0)
	Cyan	(180°,100%,100%)	#00FFFF	(0,255,255)
	Magenta	(300°,100%,100%)	#FF00FF	(255,0,255)
	Silver	(0°,0%,75%)	#BFBFBF	(191,191,191)
	Gray	(0°,0%,50%)	#808080	(128,128,128)
	Maroon	(0°,100%,50%)	#800000	(128,0,0)
	Olive	(60°,100%,50%)	#808000	(128,128,0)
	Green	(120°,100%,50%)	#008000	(0,128,0)
	Purple	(300°,100%,50%)	#800080	(128,0,128)
	Teal	(180°,100%,50%)	#008080	(0,128,128)
	Navy	(240°,100%,50%)	#000080	(0,0,128)

Converting HSV to RGB



YIQ Color Model

- **YIQ** is the color space used by the *NTSC color TV system*, employed mainly in North and Central America, and Japan.
- **I** stands for *in-phase*, while **Q** stands for *quadrature*, referring to the components used in quadrature amplitude modulation.
- In YIQ the Y component represents the ***luma*** information (*Luminance*), and is the only component used by black-and-white television receivers.
- In YIQ the **I** and **Q** stands for ***chrominance*** information.

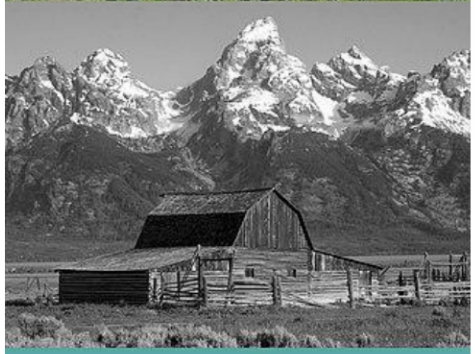
YIQ Color Model

- The primary goals of the system were to provide a signal that could be directly displayed by black and white TVs, while also providing easy coding and decoding of RGB signals.
- YIQ model is used in the conversion of grayscale images to RGB color images.
- The advantage of this model is that more bandwidth can be assigned to the Y-component (luminance) to which the human eye is more sensible than to color information.

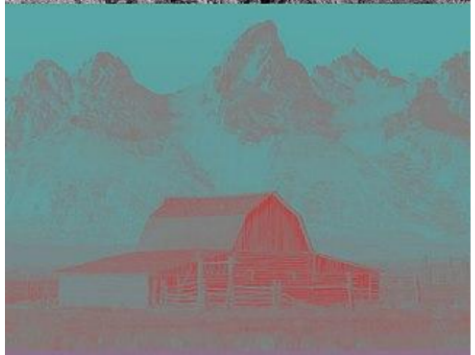
YIQ Color Model

RGB Image

Y Image

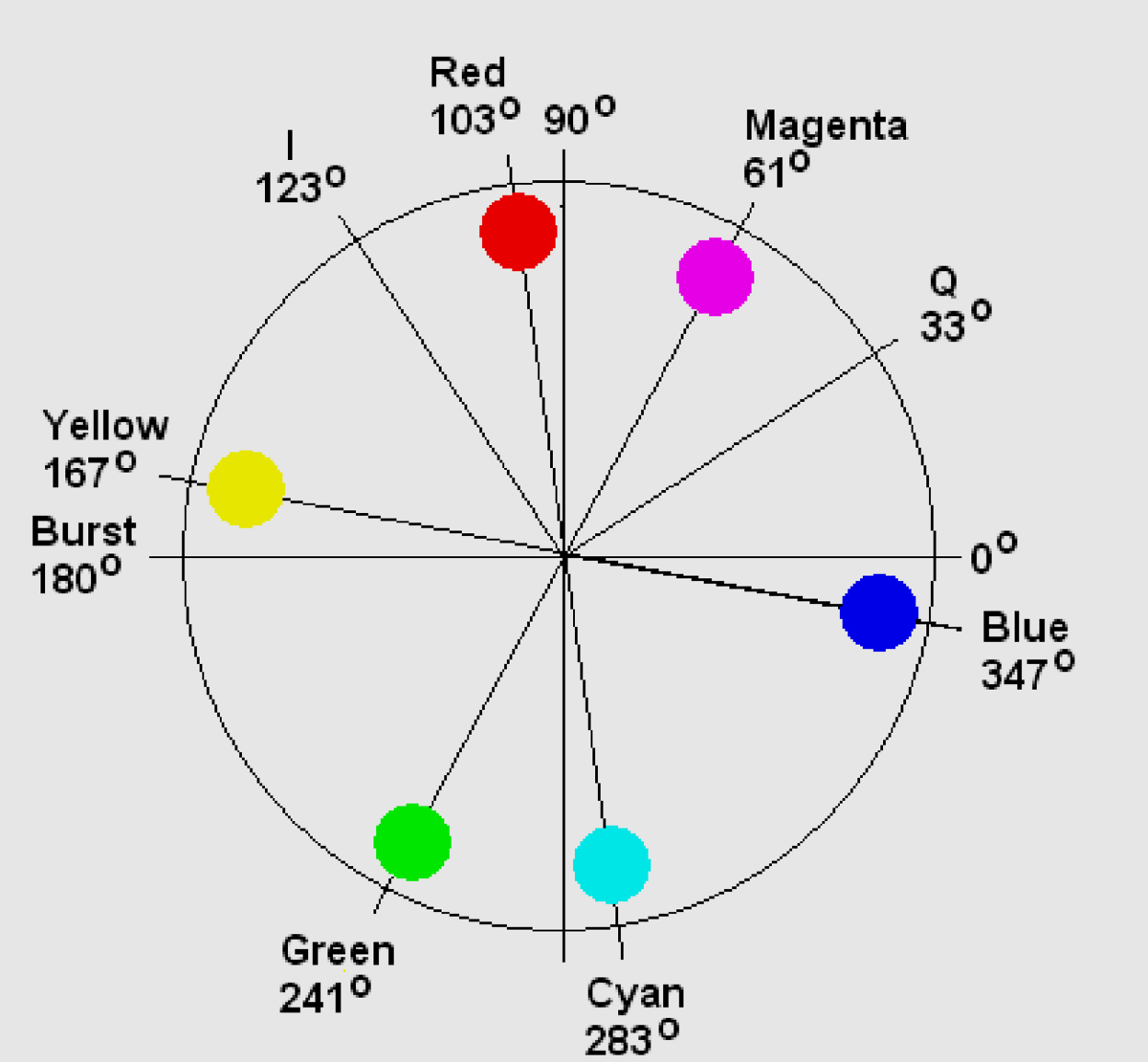


I Image

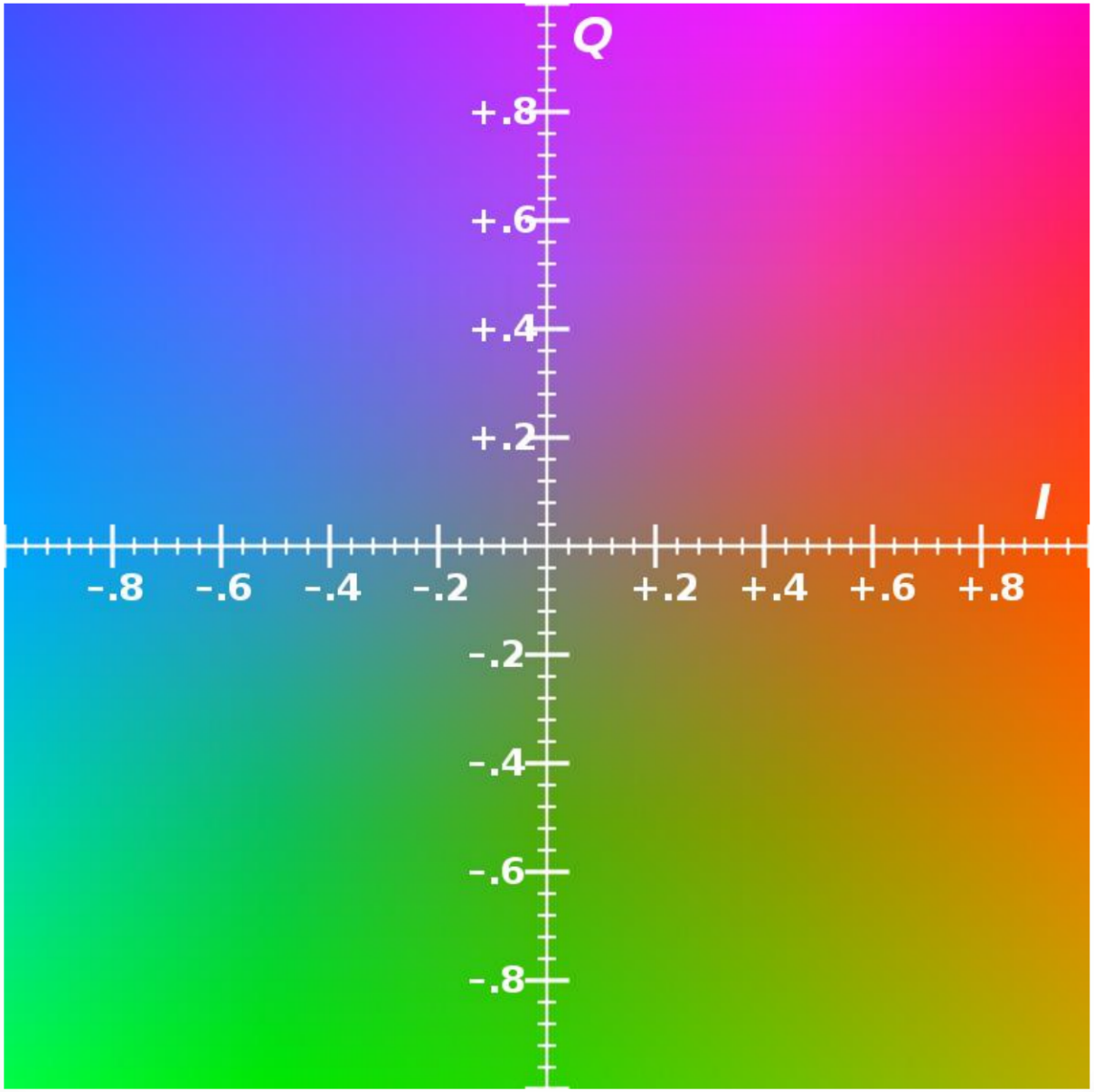


Q Image

YIQ Color Model



YIQ Color Model



Conversion between YIQ and RGB

- The conversions from RGB to YIQ and back are given by the matrices.
- There exist a formula to convert RGB into YIQ and vice-versa.

RGB to YIQ

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

YIQ to RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.621 \\ 1 & -0.272 & -0.647 \\ 1 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

where obviously the two matrices are inverses.

Conversion between YIQ and RGB

RGB to YIQ

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Y = 0.299R + 0.587G + 0.114B

I = 0.596R – 0.274G – 0.322B

Q = 0.211R – 0.523G + 0.312B

Conversion between YIQ and RGB

YIQ to RGB

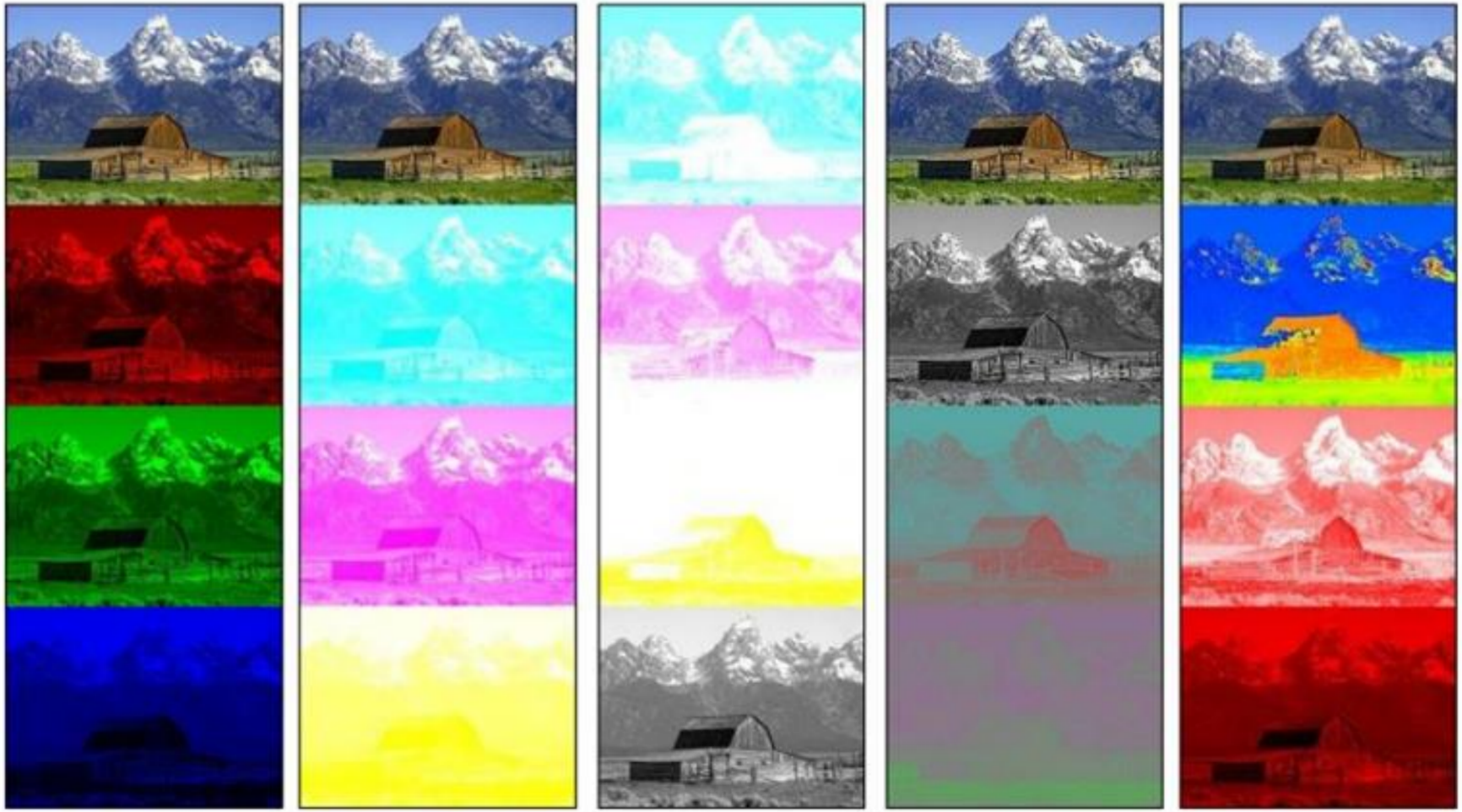
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.621 \\ 1 & -0.272 & -0.647 \\ 1 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

R = Y + 0.956I + 0.621Q

G = Y – 0.272I – 0.647Q

B = Y – 1.106I + 1.703Q

Comparison



RGB CMY CMYK YIQ HSV

Assignment

1. Draw the RGB color model cube and explain how shades of Gray are represented.
2. Explain RGB color model with its geometrical representation and mention its application area.
3. How CMY and YIQ color models differ from RGB color model? Briefly explain.
4. Explain HSV color model with its geometrical representation and mention its application area
5. Differentiate between ambient light and diffuse reflection.
6. Make distinction between specular and diffuse refection.
7. Explain CMY color model with its geometrical representation and mention its application area.

Board Exam Questions

1. What do you understand by object space and image space method? (2076 Batch)
2. Define visible surface detection. Outline the Z-buffer algorithm along with its advantages and disadvantages. (2076 Batch)
3. What do you mean by visible surface detection? Explain Z buffer algorithm for visible surface detection. (2075 Batch)
4. Write the difference between object space method and image space method. Explain Z buffer algorithm for visible surface detection. (2074 Batch)
5. Explain the scan line algorithm for visible surface detection. (2074 Batch)