3D Game Programming
2D primitive

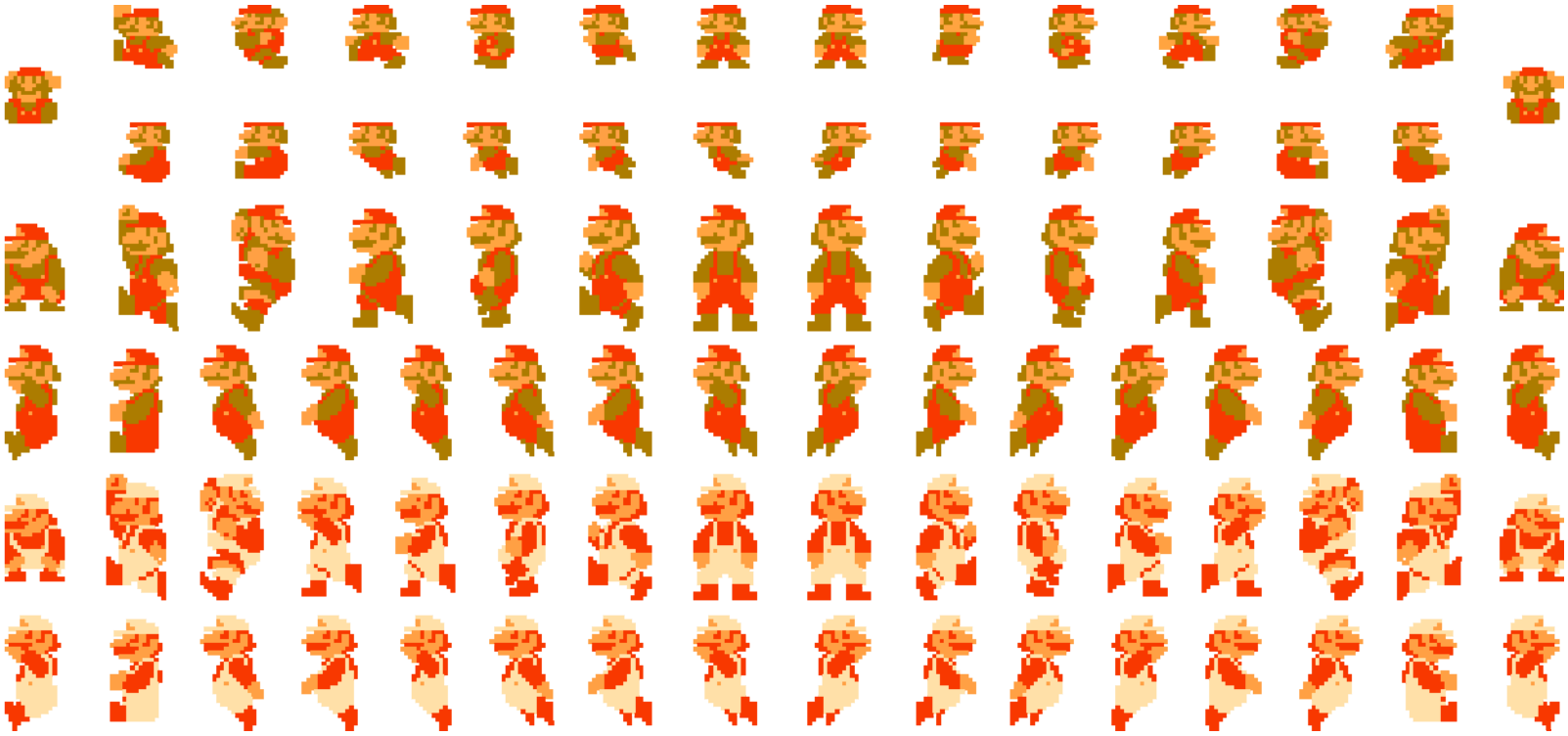
Ming-Te Chi
Department of Computer Science,
National Chengchi University

Outline

- Imaging and Raster Primitives
- Alpha and Blending
- Intersection

IMAGING AND RASTER PRIMITIVES

Sprite



Super Mario Bros. Nintendo

Electromagnetic spectrum

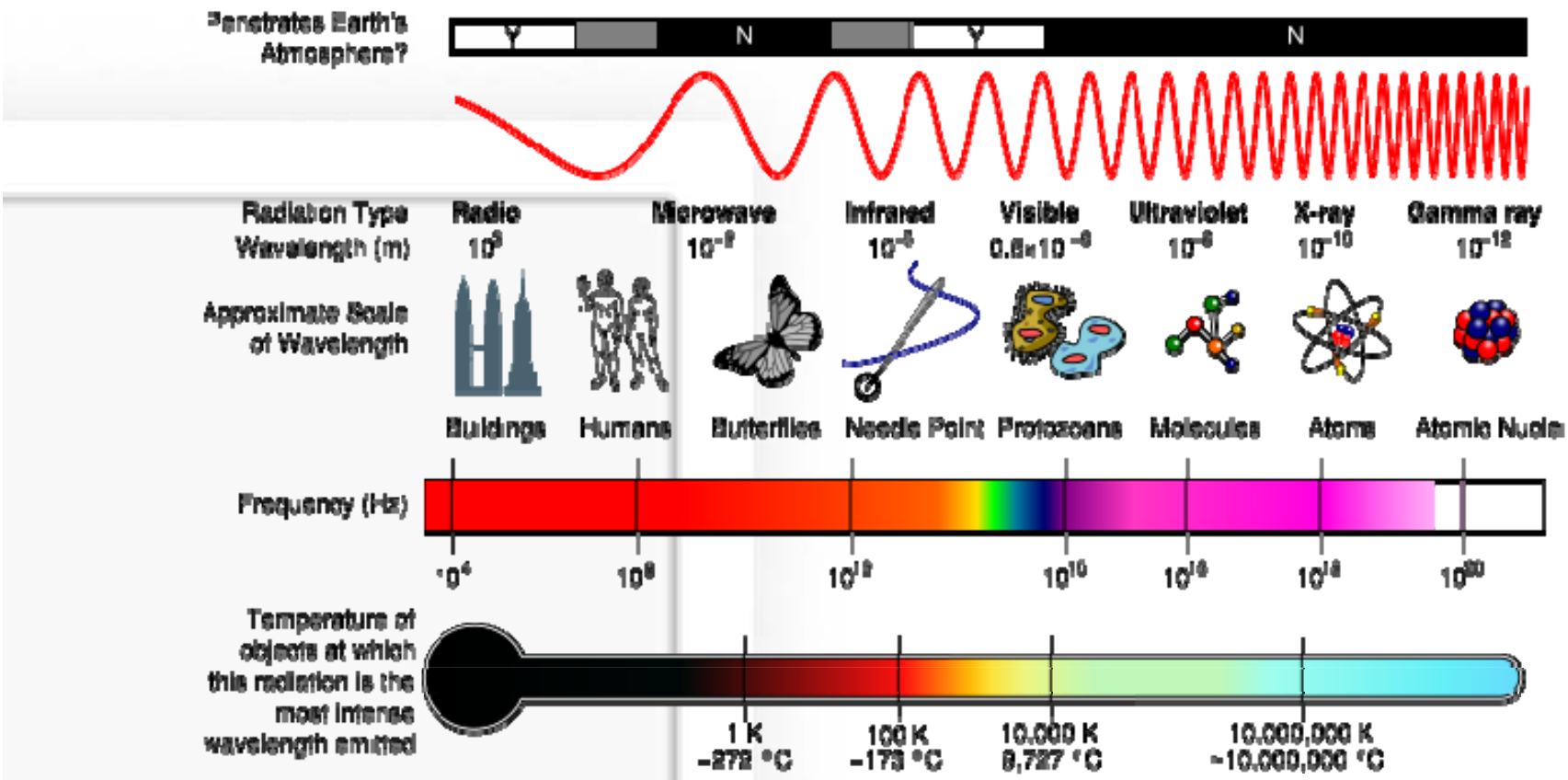
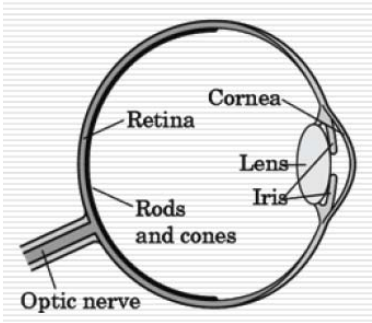
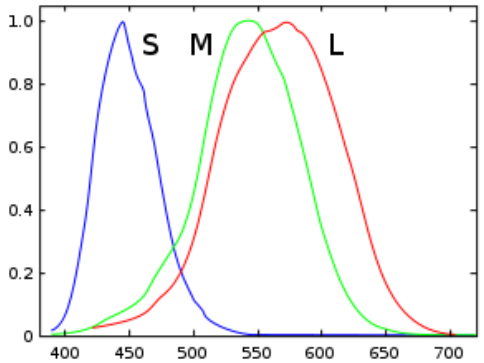


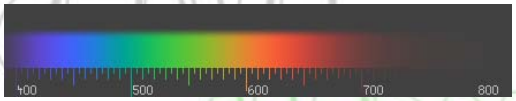
Image from wiki

Three-Color Theory

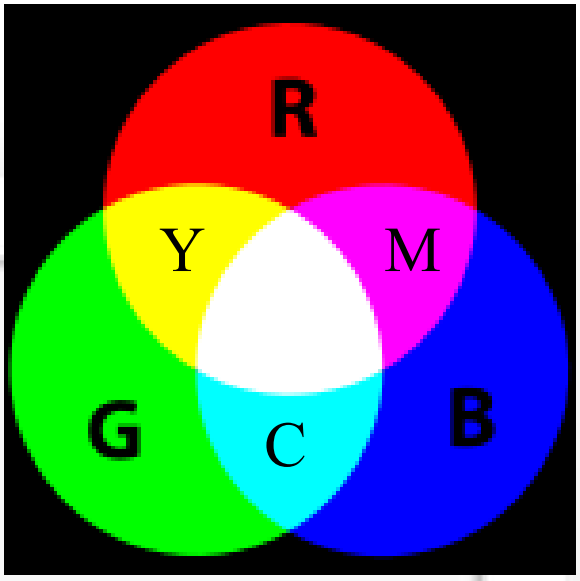
- Human visual system has two types of sensors
  - Rods:
    - monochromatic, night vision
  - Cones:
    - Color sensitive
    - Three types of cone
    - Only three values
    - (the *tristimulusvalues) are sent to the brain*





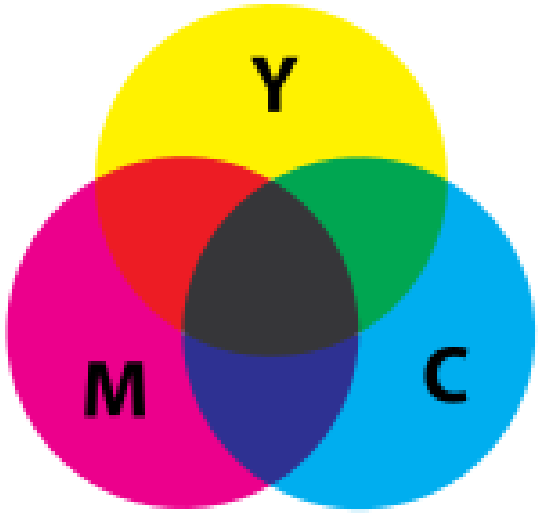


Additive / Subtractive color



Additive Color

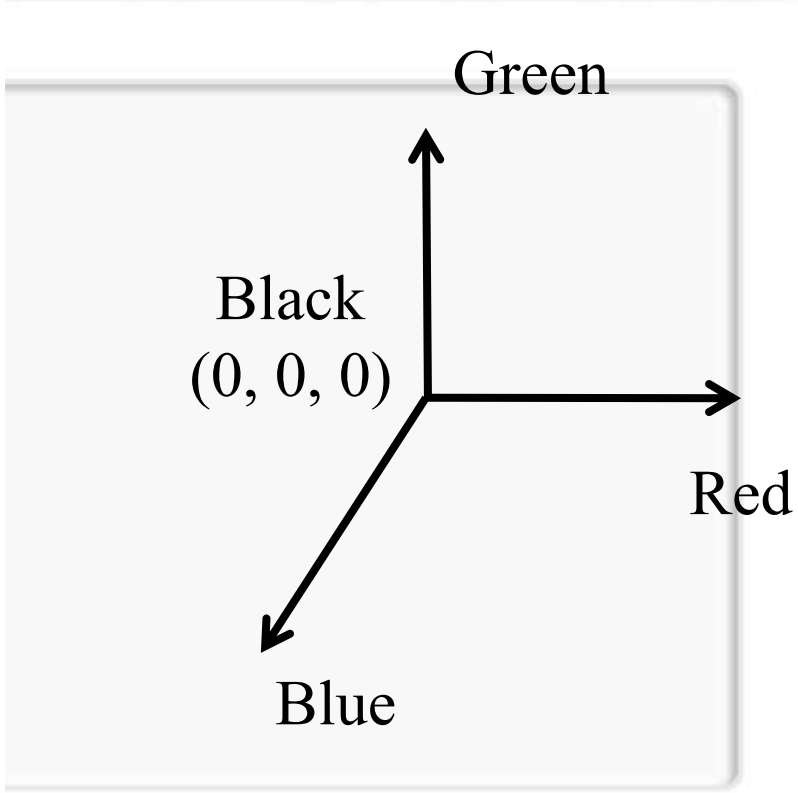
LCD, projector

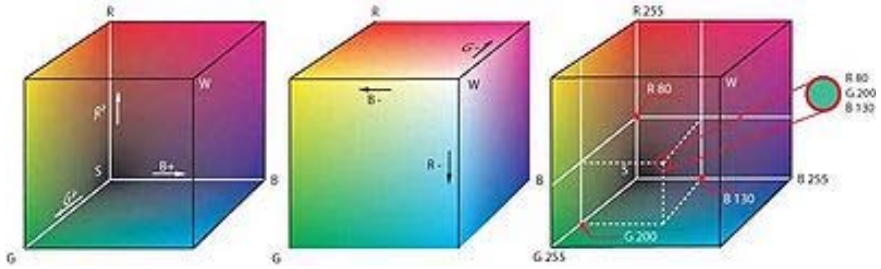


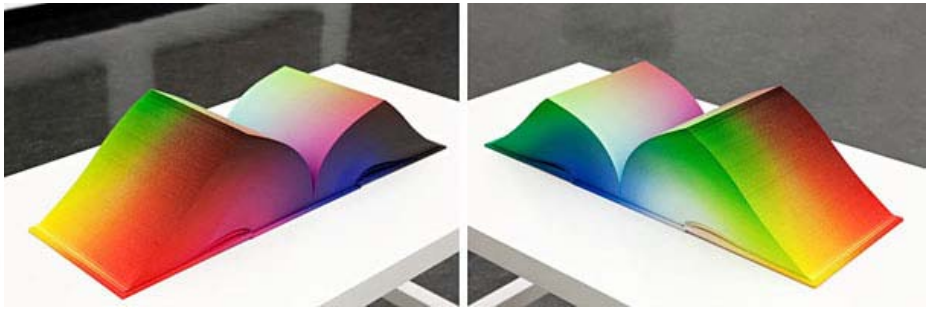
Subtractive Color

Printer

RGB color space







a 8 x 8 x 8-inch cube book
Tauba Auerbach

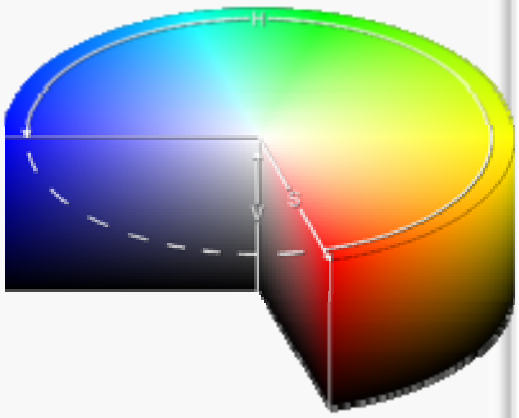
HSV color space

- *HSV*
  - hue,
  - Saturation,
  - Value,

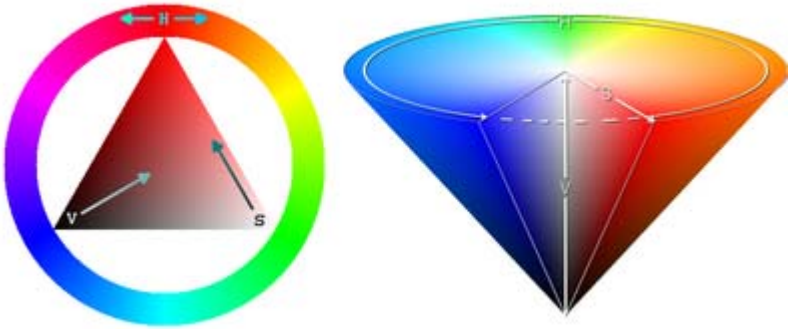
$$h = \begin{cases} 0, & \text{if } \max = \min \\ (60^\circ \times \frac{g-b}{\max - \min} + 360^\circ) \bmod 360^\circ, & \text{if } \max = r \\ 60^\circ \times \frac{b-r}{\max - \min} + 120^\circ, & \text{if } \max = g \\ 60^\circ \times \frac{r-g}{\max - \min} + 240^\circ, & \text{if } \max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } \max = 0 \\ \frac{\max - \min}{\max} = 1 - \frac{\min}{\max}, & \text{otherwise} \end{cases}$$

$$v = \max$$

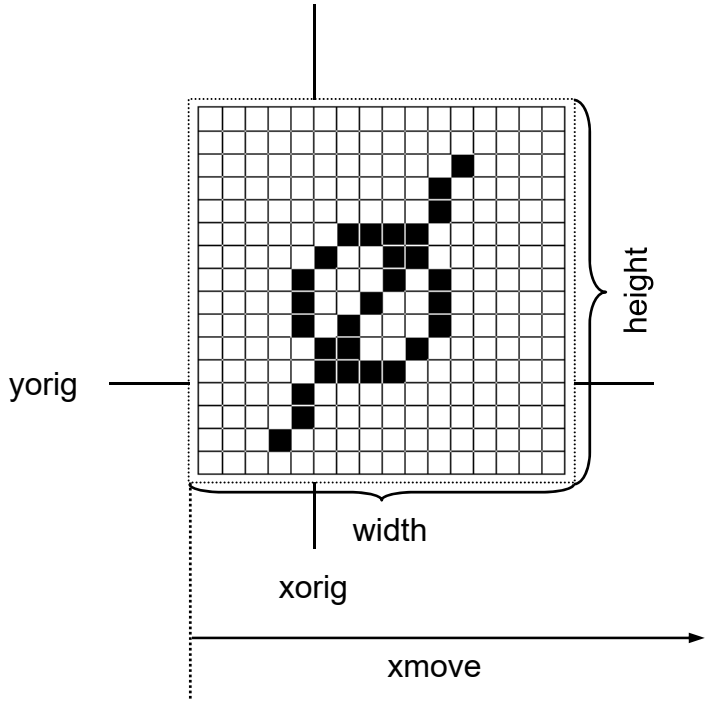


HSV



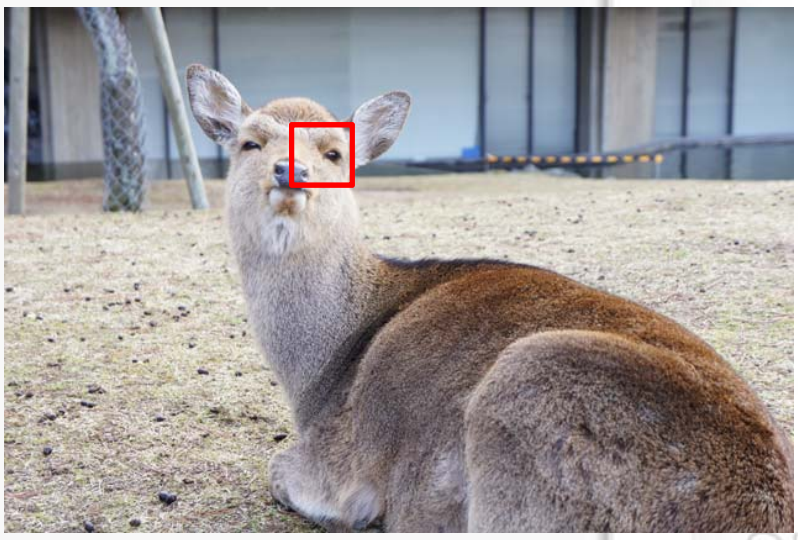
UI

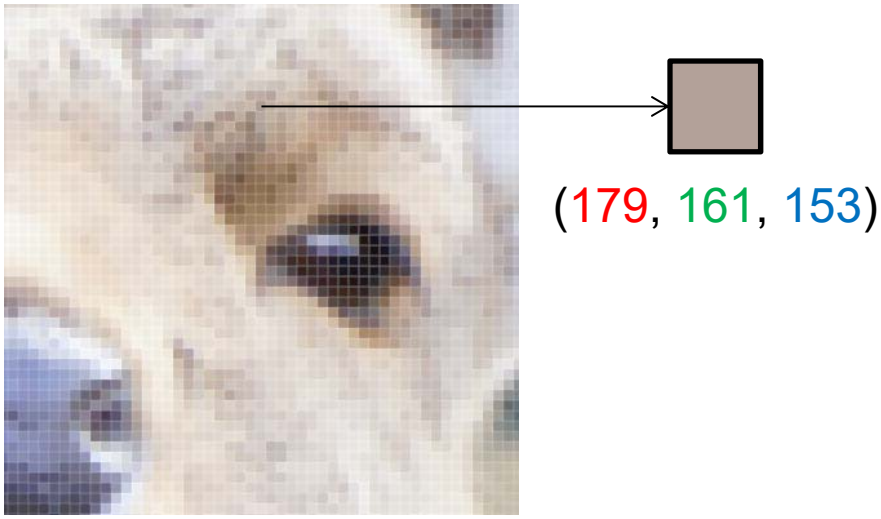
Bitmap



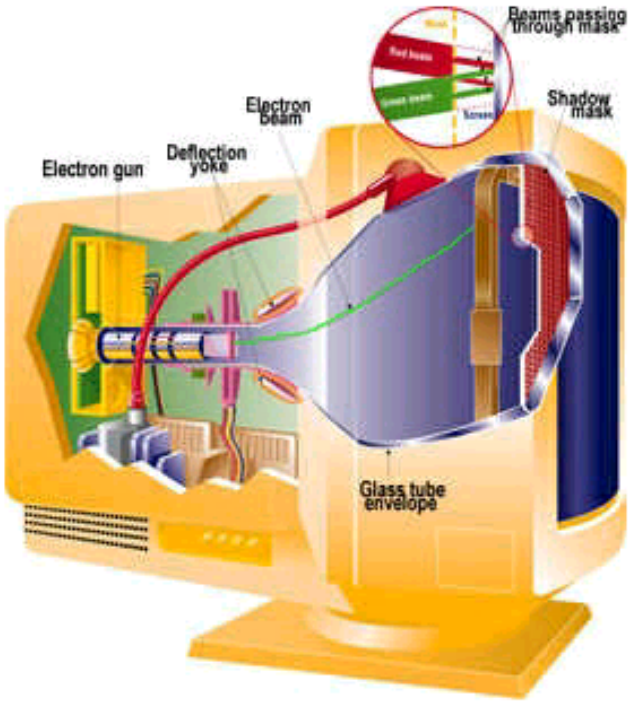
Raster Graphics

- Image produced as an array (the *raster*) of picture elements (*pixels*) in the *frame buffer*

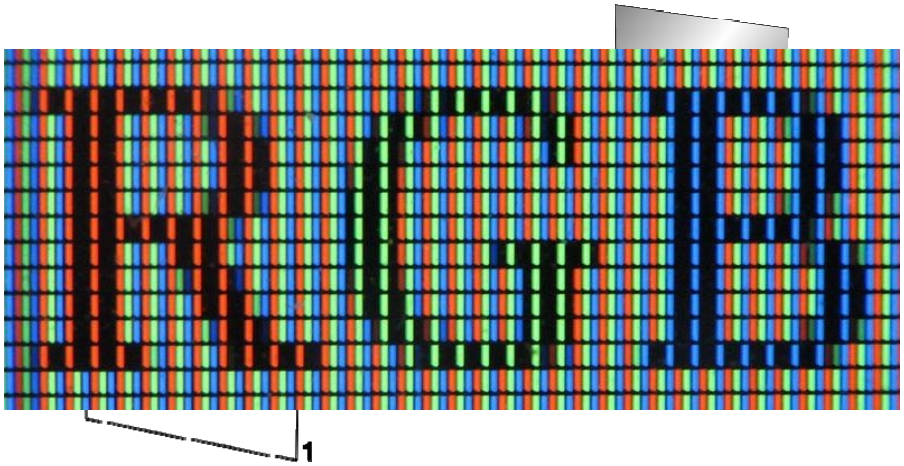




Display Technologies



CRT



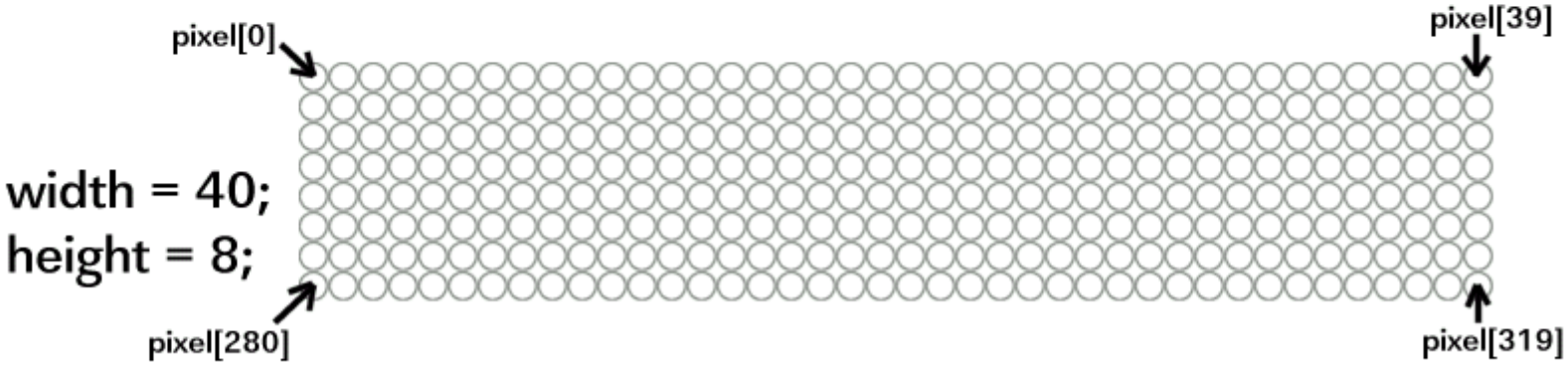
LCD

Lets Talk About Pixels

- Pixels are stored as a 1-dimensional array of *int*s
- Each *int* is formatted according to Java's standard pixel model

The 4 bytes of a 32-bit *Pixel* int.
if Alpha is 0 the pixel is transparent.
if Alpha is 255 the pixel is opaque.

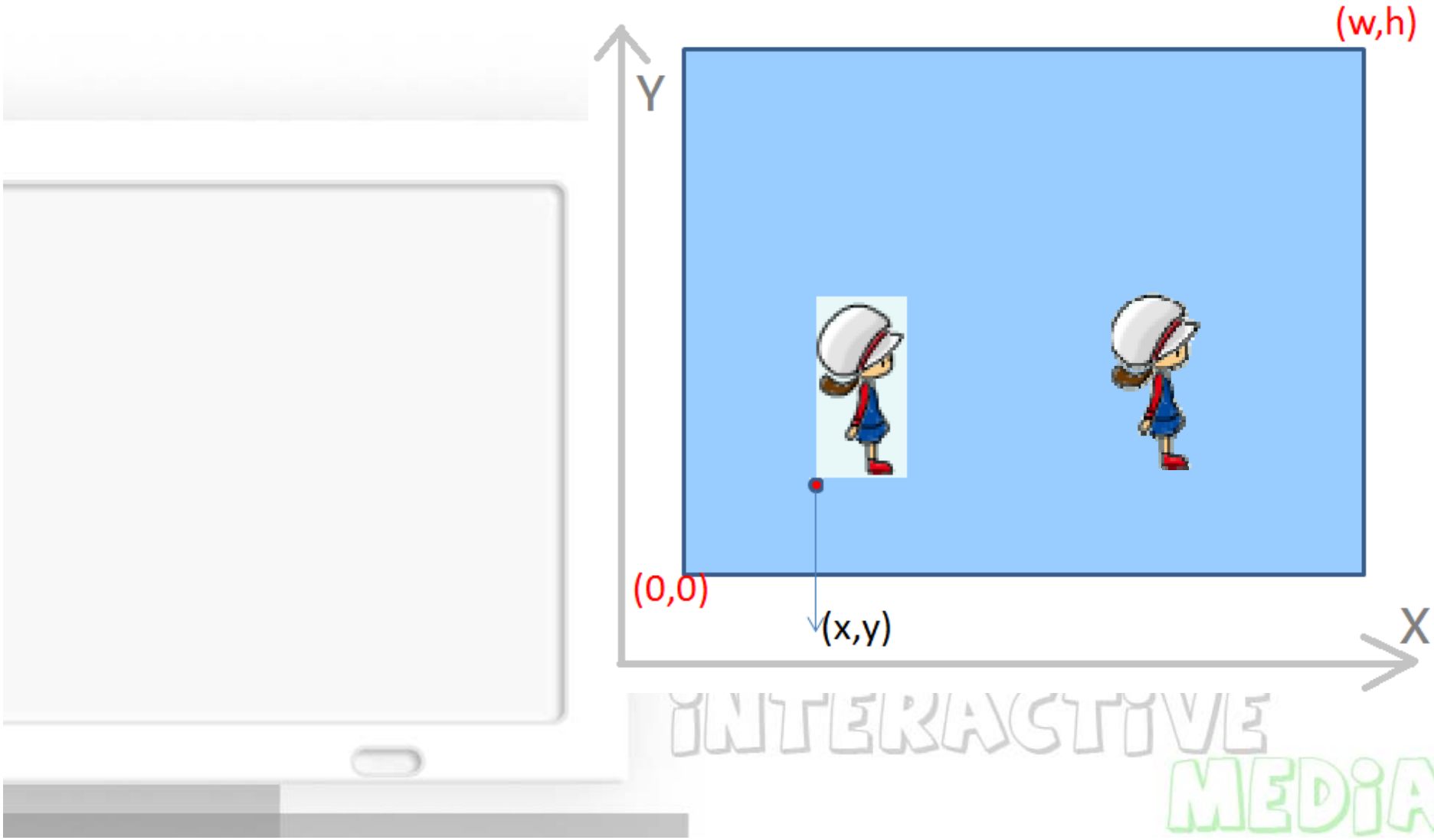
- Layout of the pixel array on the display:



- This is the image format used internally by Java

ALPHA AND BLENDING

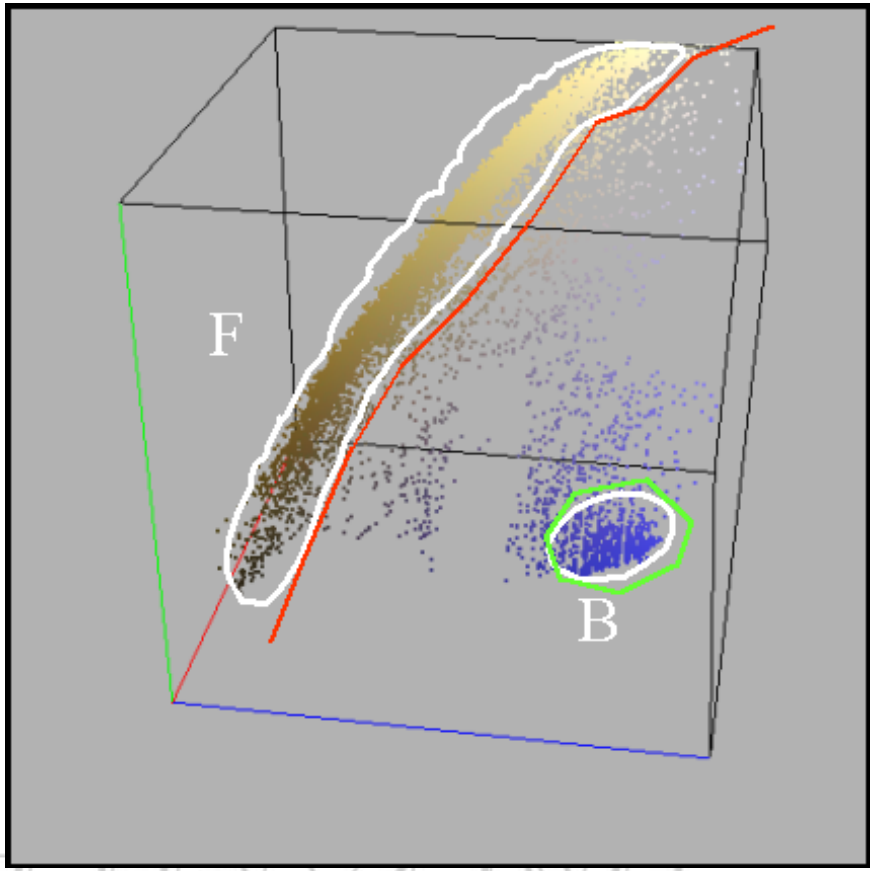
Draw image



Alpha

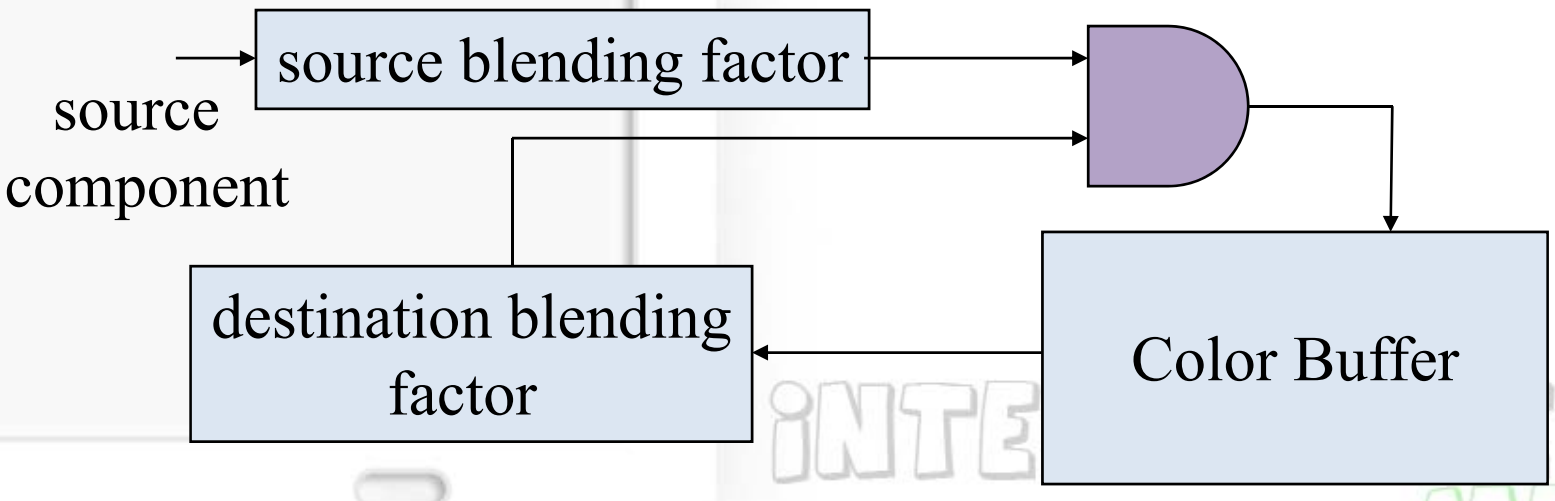
- An alpha channel, representing transparency information on a per-pixel basis.
- Alpha = 0.0f: fully transparent
- Alpha = 1.0f: fully opaque

Chroma-keying (Primatte)



Writing Model

- Use A component of RGBA (or RGBα) color to store opacity
- During rendering we can expand our writing model to use RGBA values



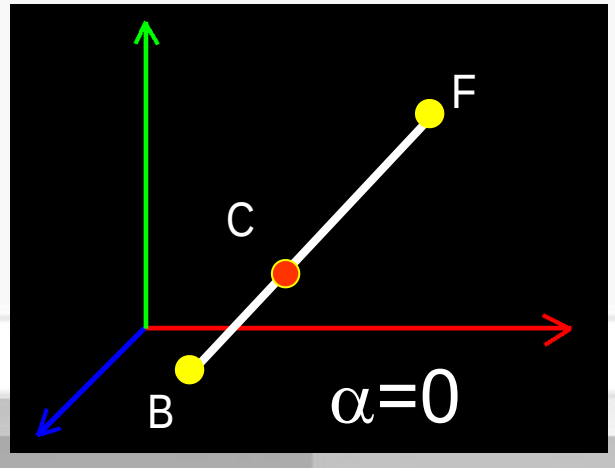
Blending

- glBlendFunc(Glenum S, Glenum D);
  - Cf = (Cs*S) + (Cd*D)
- glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

Ex: Cs={Rs, Gs, Bs, As}, Cd ={Rd, Gd, Bd, Ad},
Cf = (Cs*As) + (Cd*(1-As))

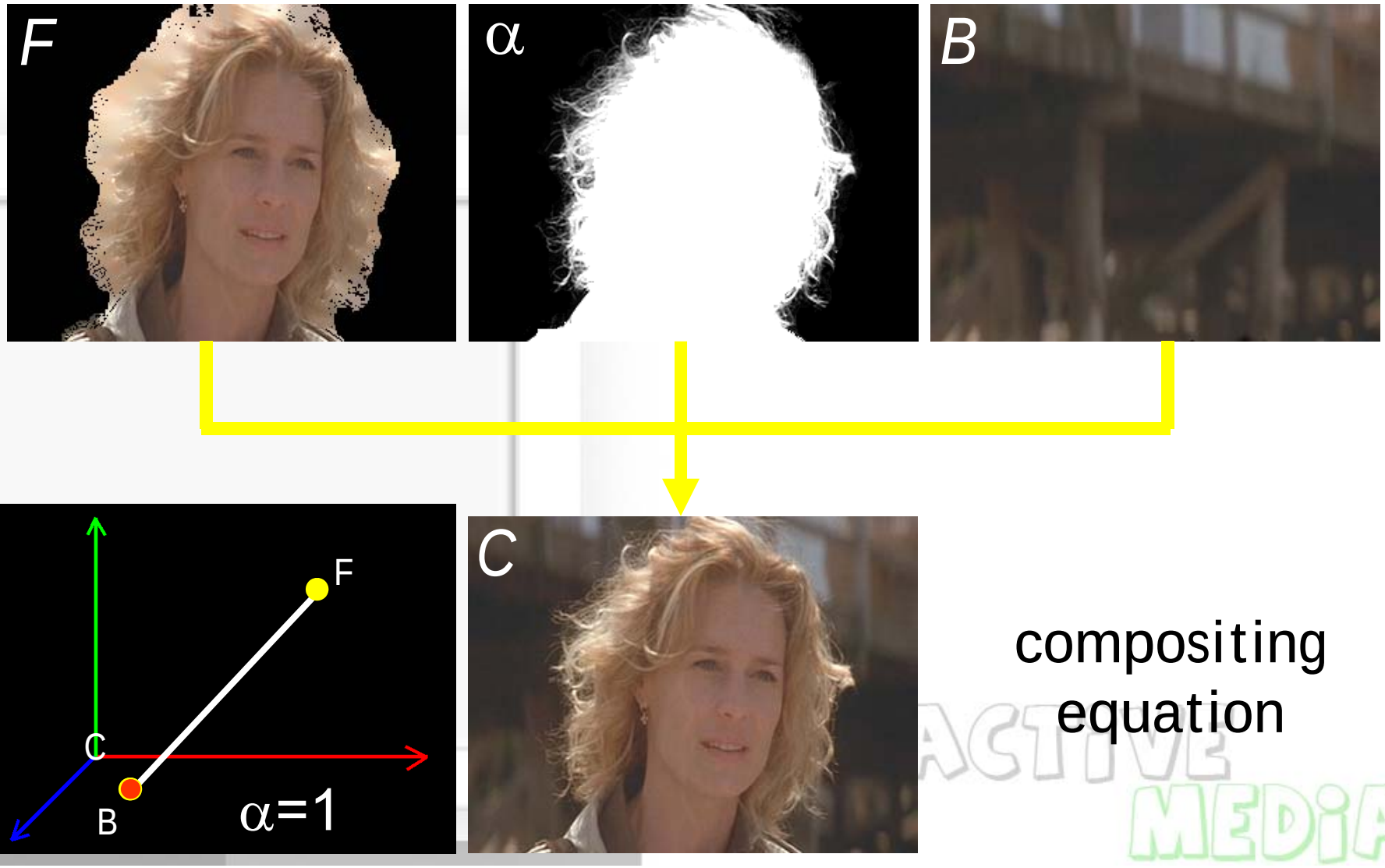
Compositing

foreground color alpha matte background



compositing equation

Compositing



compositing equation

Order Dependency

- Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
- Blending functions are order dependent

class RGBApixmap

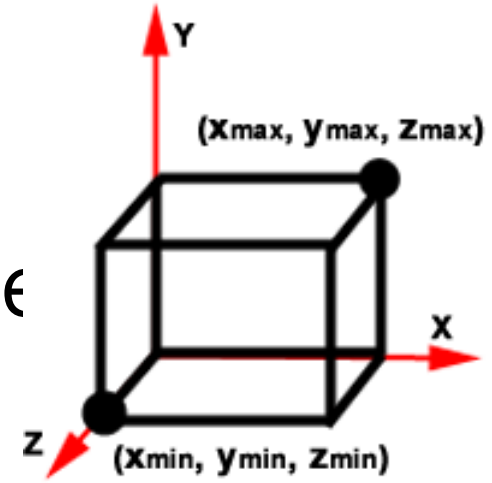
```
RGBApixmap pic;
pic.readBMPFile(  “stand.bmp”  );
pic.setChromaKey(232, 248, 248);

// draw
pic.blendtex(picX, picY, 1.0, 1.0);
```

INTERSECTION

Axis-Aligned Bounding Boxes

- Specified as two points:

$$(x_{\min}, y_{\min}, z_{\min}), (x_{\max}, y_{\max}, z_{\max})$$



- Normals are easy to calculate
- Simple point-inside test:

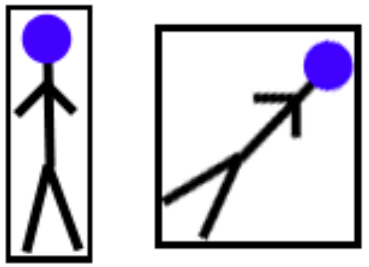
$$x_{\min} \le x \le x_{\max}$$

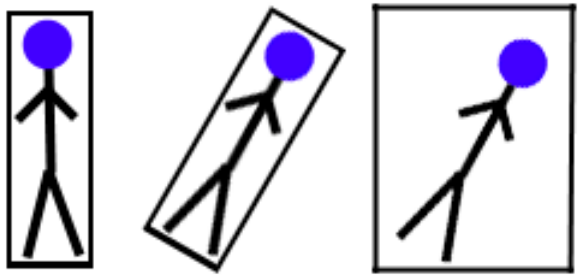
$$y_{\min} \le y \le y_{\max}$$

$$z_{\min} \le z \le z_{\max}$$

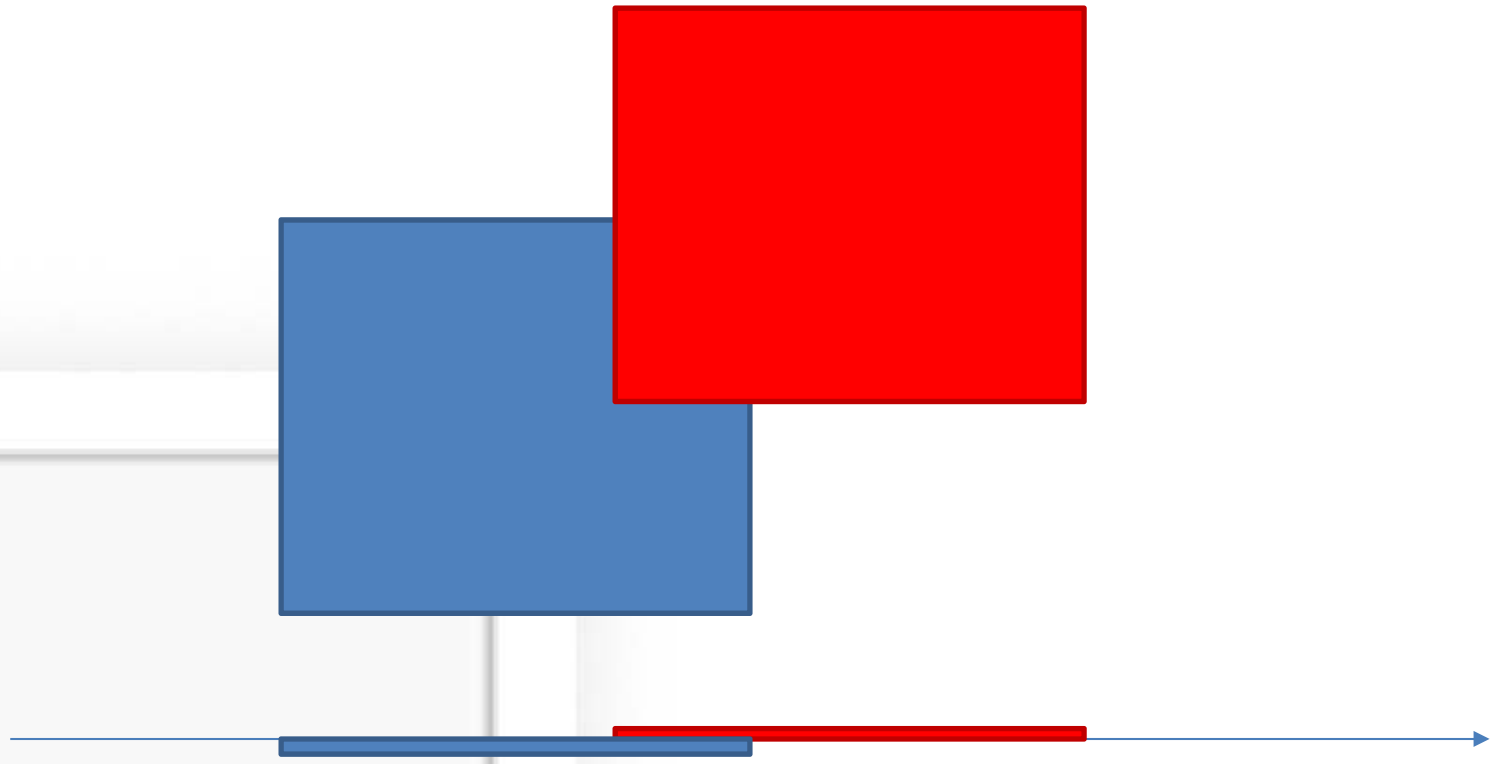
Problems With AABB's

- Not very efficient
- Rotation can be complicated
  - Must rotate all 8 points of box
  - Other option is to rotate model and rebuild AABB, but this is not efficient

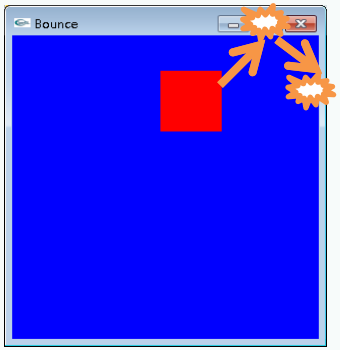





```
function isPointInsideAABB(point, box) {
  return (point.x >= box.minX && point.x <= box.maxX) &&
         (point.y >= box.minY && point.y <= box.maxY);
}
```



```
function intersect(a, b) {
        return (a.minX <= b.maxX && a.maxX >= b.minX) &&
               (a.minY <= b.maxY && a.maxY >= b.minY);
}
```



BOUNCE

Example [Bounce]

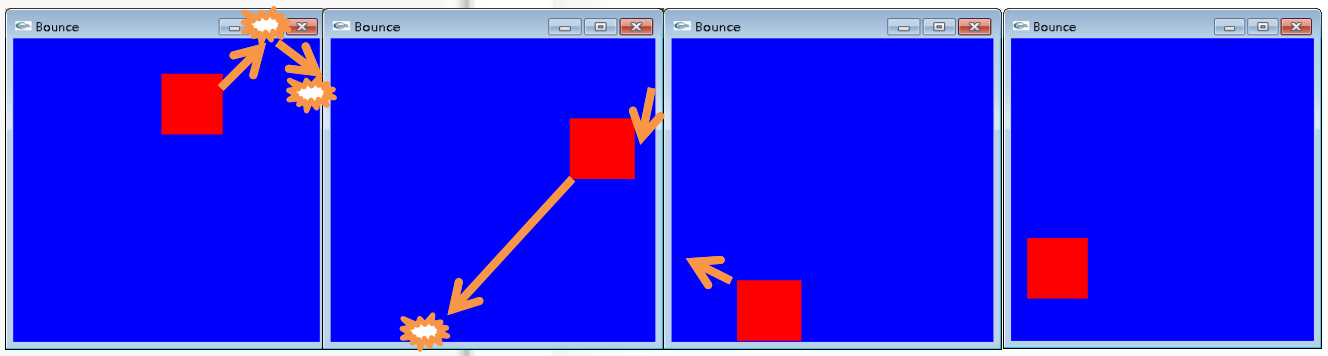
Key Function

```
void glutTimerFunc
(unsigned int msecs, void(*func)(int value), int value);
```

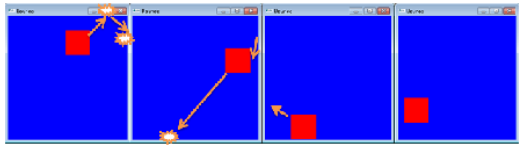
*Registers a timer callback to be triggered in a specified number of milliseconds.

- *msecs* : Number of milliseconds to pass before calling the callback.
- *func* : The timer callback function.
- *value* : Integer value to pass to the timer callback.

Bounce animation



...



```
void TimerFunction(int value)
{
        // Reverse direction left 、right 、top 、bottom edge
        if(x1>windowWidth-rsize ||  x1<0) xstep=-xstep;
        if(y1>windowHeight-rsize ||  y1<0)  ystep=-ystep;

        //Check bounds
        if(x1>windowWidth-rsize) x1=windowWidth-rsize-1;
        if(y1>windowHeight-rsize) y1=windowHeight-rsize-1;

        //Actually move the square
        x1+=xstep;
        y1+=ystep;

        //Redraw the scene with new coordinates
        glutPostRedisplay();
        glutTimerFunc(33,TimerFunction,1);
        //self recall per 33msecs.
}
```