

Chapter Six

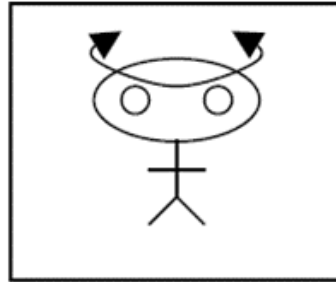
Animation and Surface Rendering

Introduction

- Animation is the creation of the “illusion of movement” using a series of still images.
- Persistence of vision refers to brain retaining the image of what eyes see even after the image is no longer visible.
- The brain can only process a certain number of images at a time.
- Brain can recognize images as separate images if they are viewed at 12 or fewer images per second.
- If the pictures appear faster than 12 per second they begin to merge into each other creating the illusion of movement.
- Television and movies are usually created at 24 to 30 images per second

The Fundamental Principles of Animation

- The principles are:
 - Timing:** Timing is the essence of animation. The speed at which something moves gives a sense of what the object is, the weight of an object, and why it is moving. Something like an eye blink can be fast or slow. If it's fast, a character will seem alert and awake. If it's slow the character may seem tired and lethargic.
- **Example:** head that turns left and right



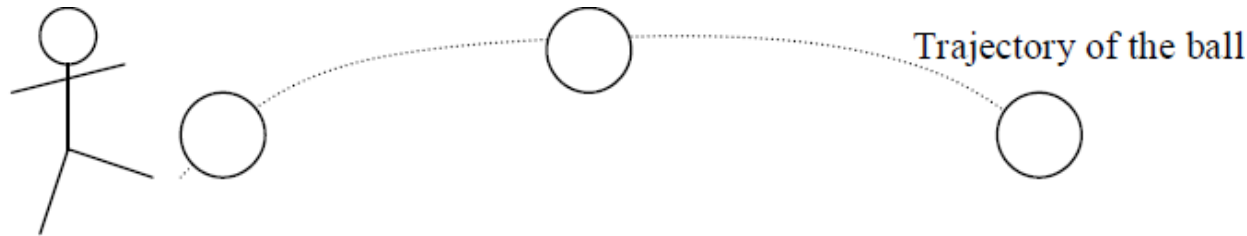
- Head turns back and forth really slow: it may seem as if the character is stretching his neck (lots of in between frames).
- A bit faster it can be seen as saying "no" (a few in between frames)
- Really fast, and the character is reacting to getting hit by a baseball bat (almost none in between frames).

ii. Ease In and Out (or Slow In and Out):

- Ease in and out has to do with gradually causing an object to accelerate, or come to rest, from a pose.
- An object or limb may slow down as it approaches a pose (Ease In) or gradually start to move from rest (Ease Out). For example, a bouncing ball tends to have a lot of ease in and out when at the top of its bounce.
- As it goes up, gravity affects it and slows down (Ease In), then it starts its downward motion more and more rapidly (Ease Out), until it hits the ground.

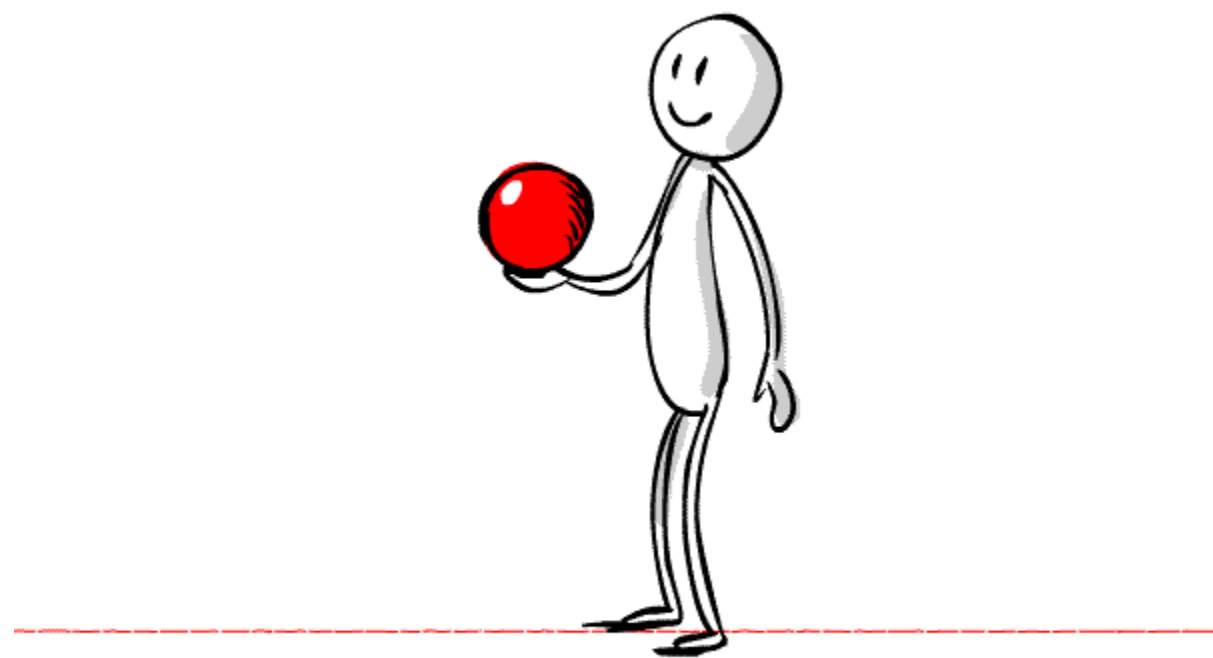
iii. Arcs:

- In the real world almost all action moves in an arc.
- When creating animation one should try to have motion follow curved paths rather than linear ones.
- It is very seldom that a character or part of a character moves in a straight line.
- Even gross body movements when you walk somewhere tend not be perfectly straight.
- When a hand/arm reaches out to reach something, it tends to move in an arc.
- Simple example – Kicking a ball



iv. Anticipation:

- Action in animation usually occurs in three sections.
- The setup for the motion, the actual action and then follow-through of the action.
- The first part is known as anticipation. In some cases anticipation is needed physically.
- For example, before you can throw a ball you must first swing your arm backwards. The backwards motion is the anticipation; the throw itself is the motion. Anticipation is used to lead the viewers eye to prepare them for the action that follows. Longer period of anticipation is needed for faster actions.

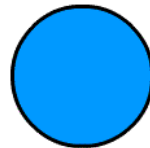


v. Exaggeration:

- Exaggeration is used to accent an action.
- It should be used in a careful and balanced manner, not arbitrarily. Figure out what the desired goal of an action or sequence is and what sections need to be exaggerated. The result will be that the animation will seem more realistic and entertaining. One can exaggerate motions, for example an arm may move just a bit too far briefly in an extreme swing.

vi. Squash and Stretch:

- Squash and stretch is a way of deforming an object such that it shows how rigid the object is.
- For example if a rubber ball bounces and hits the ground it will tend to flatten when it hits. This is the squash principle. As it starts to bounce up it will stretch in the direction it is going.



vii. Secondary Action:

- Secondary action creates interest and realism in animation.
- It should be staged such that it can be noticed but still not overpower the main action.
- A good example of this is a character at a table acting and delivering their main acting. A side piece of acting business might be the character thumbing their fingers on the table. This isn't the main action say, perhaps it occurs as the other hand is more largely gesturing and your focus is on the face.

viii. Follow Through and Overlapping Action:

- Follow Through is the same as anticipation, only at the end of an action.
- It is usually animated as something goes past its resting point and then coming back to where it would normally be.
 - For example, in throwing a ball, you put your hand back, that's anticipation; it's the preparation for the throwing action itself. Then you throwing arm comes forward for the main action. Follow Through is then the arm continuing past the normal stopping point, overshooting it and then coming back. The arm has continued or "followed through" on the action it was doing before returning back to rest.
- Overlapping Action is an action that occurs because of another action.
 - For example if a dog is running and suddenly comes to a stop, its ears will probably still keep moving for a bit. Another example, if an alien is walking and it has an antenna on it, the antenna will probably sway as a result of the main body motion. This is overlapping action. It is caused because of the main motion and overlaps on top of the main motion.

ix. Straight Ahead Action and Pose-To-Pose Action:

- There are 2 basic methods to creating animation.
- Straight ahead animation is one where the animator draws or sets up objects one frame at a time in order.
 - For example, the animator draws the first frame of the animation, then draws the second, and so on until the sequence is complete. In this way, there is one drawing or image per frame that the animator has setup. This approach tends to yield a more creative and fresh look but can be difficult to time correctly and tweak.
- The other approach is Pose-To-Pose animation. Pose to Pose is created by drawing or setting up key poses and then drawing or creating in-between images. This is the basic computer "key frame" approach to animation. It is excellent for tweaking timing and planning out the animation ahead of time. You figure out the key poses, and then the motion in-between is generated from that. This is very useful when specific timing or action must occur at specific points.

x. Staging:

- Staging is presenting an action or item so that it is easily understood.
- An **action** is staged so that it is understood; a **personality** is staged so that it is recognizable; an **expression** so that it can be seen; a **mood** so that it will affect the audience.
- In general, it is important that action is presented one item at a time. Background characters must be animated such that they are still "alive", but not so much that they steal the viewer's attention from the main action.

xi. Appeal:

- Appeal means anything that a person likes to see. Appeal can be gained by correctly utilizing other principles such as exaggeration in design, avoiding symmetry, using overlapping action, and others.

xii. Personality:

- Personality determines the success of an animation. The idea is that the animated creature really becomes alive and enters the true character of the role.

Techniques of Animation

- Historically there are 3 major types of animation:

1. Hand Drawn Animation

- Done by an artist who draws each character and movement individually
- Very time consuming to have to draw, then color, then photograph each picture
- Draw pictures first, then color them on celluloid, then they take pictures and animate them
- Very expensive due to hours of labor involved

2.Stop Motion Animation

- Can be done by virtually anyone, with no extensive training
- Does not take that much time relative to the other 2 methods
- Uses jointed figures or clay figures that can be moved to make motions
- Take still pictures of the individual movements, then use relatively inexpensive computer software to animate
- We use Movie Maker Software to complete our animations
- Not very expensive because all you need is a digital camera and the software to animate



3. Computer Animation

- All characters and movements are generated using computer animation software
- Can also be very time consuming as they can get very complicated in movements and effects
- All characters are fully animated with no still pictures
- Can be very expensive because of the complexity of the stunts and animations being done
- Huge budgets because the animation sequences more complicated these days

Sequences of steps to produce full Animation

- **Step 1**

- The initial step to creating a computer animation is to conceptualize the story. This mental image creation process is important because it defines the flow of movement that will be observed by the objects in the animation. It will also define the starting, middle, and ending position of all objects concerned allowing the user to identify where the looping process begins.

- **Step 2**

- The background must then be introduced into the frame. In general, anything that does not require movement or is not defined as a character is considered as a background element. So if you are creating an animation of a butterfly moving from one flower to another under a cloudy sky, the flowers are part of the background. If the clouds are moving, then they are considered as characters just like the butterfly.

- **Step 3**

- Proceed by drawing every pose that you intend the characters to assume. Each pose must be made on a separate drawing . The drawings must reflect the changes in the character's appearance like a butterfly with closed wings, open wings, touching a flower, flying between the flowers, and so on. The number of drawings would depend on the number of poses the characters would assume. Like in the example, the butterfly may need four different drawings while the clouds may only need two.

- **Step 4**

- Make multiple copies of the background image based on the number of drawings required in the animation. Merge the initial pose of the characters with the background to create the initial frame. In the example, only two drawings of the clouds were made. Each drawing must have its own copy to correspond with the four drawings of the butterfly. In the same manner, there must be four copies of the background image.



- **Step 5**

- Save each frame with a different file name. To make sure that there is continuity and that the frame will flow seamlessly, copy the previous frame into the new frame deleting only the pose of the characters and replacing them with the new pose.

- **Step 6**

- Repeat Step 5 for all remaining frames until all the poses have been completed. Remember to save each frame with a different file name and take care not to overwrite previous frames.

Step 7

- When all of these are finished, it is necessary to load animation packages with rendering engines like the GIF Construction set, Swish, Anim8or, Blender 3D, iMovie, or Windows Movie Maker among others. The user must load the frames in the order which they will be logically displayed. The delay in the display of the frames will dictate how fast or slow the movement will be in the final animated file.

Step 8

- The completed file is usually saved in the GIF format. The output may also be passed through a compression package to minimize its size for use with websites or easier distribution.

- In OpenGL we can animate our graphics using a timer function or `glutIdleFunc()`

Hidden Surface Removal

- When you draw a scene composed of three-dimensional objects, some of them might obscure all or parts of others.
- Changing your viewpoint can change the obscuring relationship.
 - For example, if you view the scene from the opposite direction, any object that was previously in front of another is now behind it.
- To draw a realistic scene, these obscuring relationships must be maintained.

- The elimination of parts of solid objects that are obscured by others is called hidden-surface removal.
- Methods can be categorized as:
 - Object Space Methods
 - These methods examine objects, faces, edges etc. to determine which are visible. The complexity depends upon the number of faces, edges etc. in all the objects.
 - Image Space Methods
 - These methods examine each pixel in the image to determine which face of which object should be displayed at that pixel. The complexity depends upon the number of faces and the number of pixels to be considered.

Z Buffer

- The easiest way to achieve hidden-surface removal is to use the depth buffer (sometimes called a z-buffer).
- A depth buffer works by associating a depth, or distance from the viewpoint, with each pixel on the window.
- Initially, the depth values for all pixels are set to the largest possible distance, and then the objects in the scene are drawn in any order.
- Graphical calculations in hardware or software convert each surface that's drawn to a set of pixels on the window where the surface will appear if it isn't obscured by something else.

- In addition, the distance from the eye is computed.
- With depth buffering enabled, before each pixel is drawn, a comparison is done with the depth value already stored at the pixel.
- If the new pixel is closer to the eye than what is there, the new pixel's color and depth values replace those that are currently written into the pixel.
- If the new pixel's depth is greater than what is currently there, the new pixel would be obscured, and the color and depth information for the incoming pixel is discarded.
- Since information is discarded rather than used for drawing, hidden-surface removal can increase your performance.

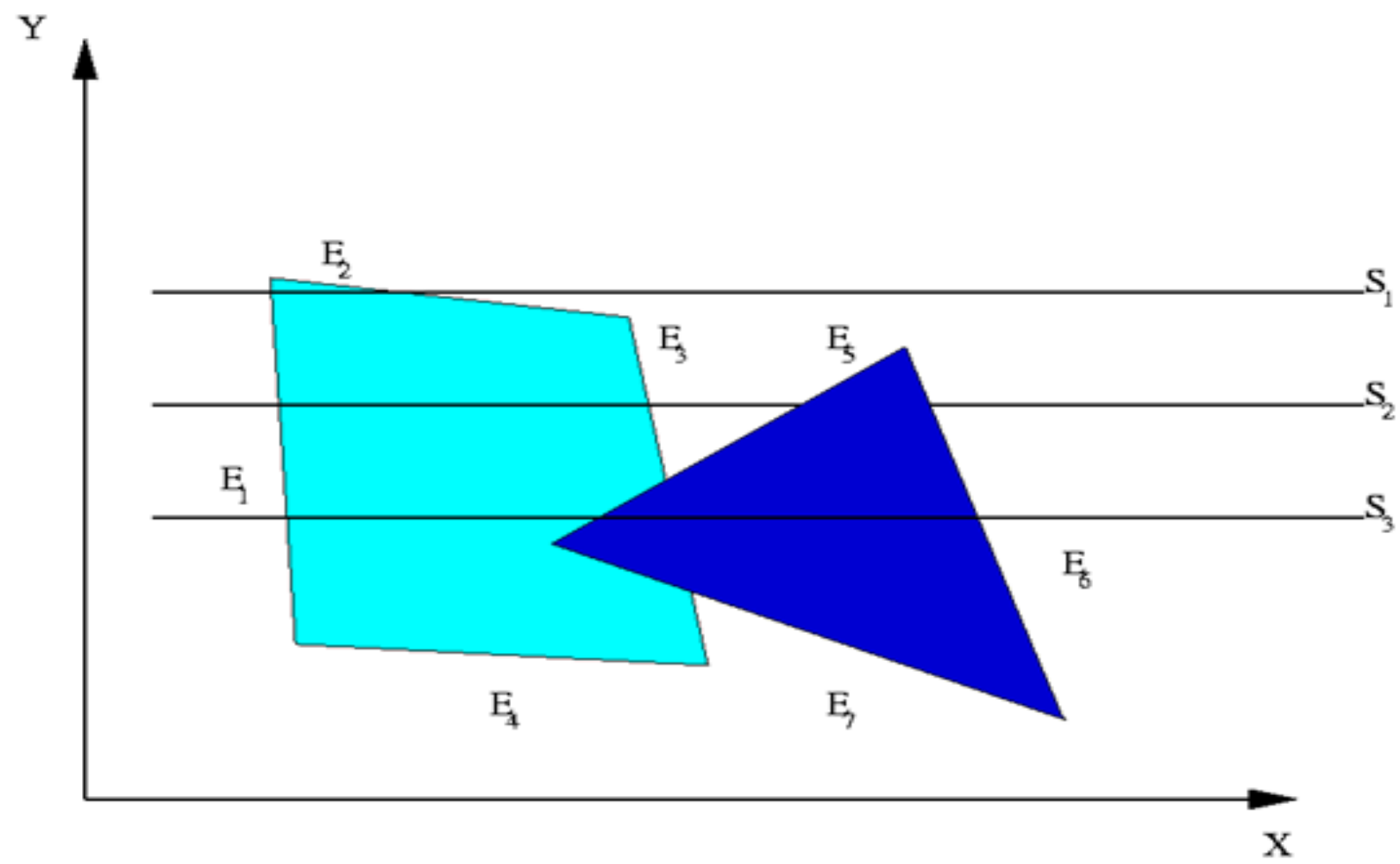
Z Buffer in OpenGL

- To use depth buffering in OpenGL, you need to enable depth buffering. This has to be done only once.
- Each time you draw the scene, before drawing you need to clear the depth buffer and then draw the objects in the scene in any order.

Scan-Line Algorithm

- The scan-line algorithm is another image-space algorithm.
- It processes the image one scan-line at a time rather than one pixel at a time.
- By using area coherence of the polygon, the processing efficiency is improved over the pixel oriented method.
- Using an active edge table, the scan-line algorithm keeps track of where the projection beam is at any given time during the scan-line sweep.
- When it enters the projection of a polygon, an IN flag goes on, and the beam switches from the background color to the color of the polygon

- After the beam leaves the polygon's edge, the color switches back to background color.
- To this point, no depth information need be calculated at all.
- However, when the scan-line beam finds itself in two or more polygons, it becomes necessary to perform a z-depth sort and select the color of the nearest polygon as the painting color.



Reading assignment

- painter's algorithm
- Back face removal algorithm

Light Sources

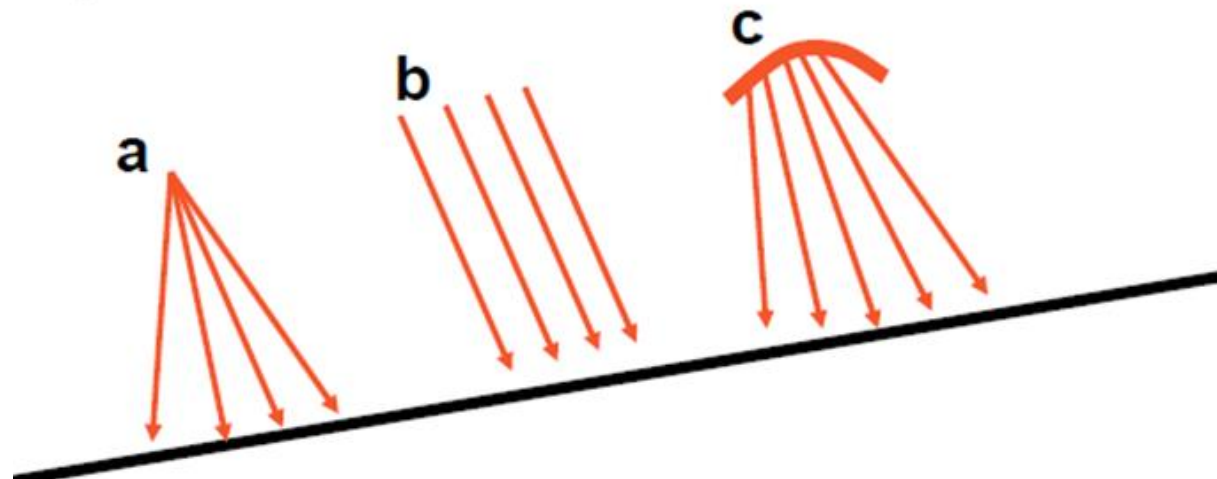
- Lighting refers to the handling of interactions between the light sources and the objects in the 3D scene.
- Lighting is one of the most important factor in producing a realistic scene.
- The color that we see in the real world is the result of the interaction between the light sources and the surfaces color of the material.
 - In other words, three parties are involved: **viewer**, **light sources**, and **the material**.
- When light (of a certain spectrum) from a light source strikes a surface, some gets absorbed, some is reflected or scattered.
- The angle of reflection depends on the angle of incidence and the surface normal.

- The amount of scatterness depends on the smoothness and the material of the surface.
- The reflected light also spans a certain color spectrum, which depends on the color spectrum of the incident light and the absorption property of the material.
- The strength of the reflected light depends on the position and distance of the light source and the viewer, as well as the material.
- The reflected light may strike other surfaces, and some is absorbed and some is reflected again.
- The color that we perceived about a surface is the reflected light hitting our eye.
- In a 2D photograph or painting, objects appear to be three-dimensional due to some small variations in colors, known as **shades**.

- There are two classes of lighting models:
 1. **Local illumination:** consider only the direct lightings. The color of the surface depends on the reflectance properties of the surface and the direct lightings.
 2. **Global illumination:** in real world, objects received indirect lighting reflected from other objects and the environment. The global illumination model considers indirect lightings reflected from other objects in the scene. Global illumination model is complex and compute intensive.
- In order to produce realistic images, we must simulate the appearance of surfaces under various lighting conditions.

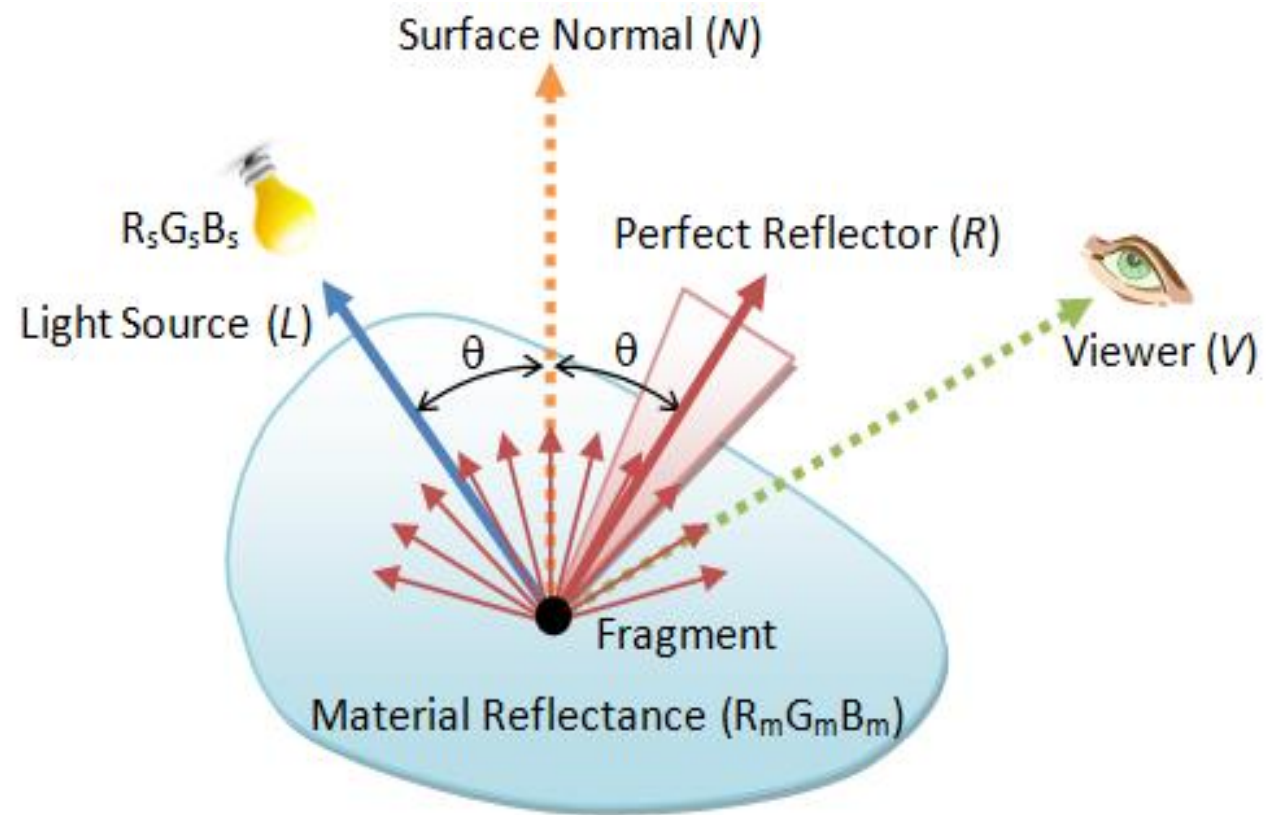
Light Source Models

- **Point Source (a):** All light rays originate at a point and radially diverge. A reasonable approximation for sources whose dimensions are small compared to the object size
- **Parallel source (b):** Light rays are all parallel. May be modeled as a point source at infinite distance (the sun)
- **Distributed source (c):** All light rays originate at a finite area in space. It models a nearby source, such as a fluorescent light



The Phong model

- There are many physically based models of how light interacts with materials.
- Unfortunately, these models are too complex for real time graphics
- Phong propped an approximate lighting model that is easy to compute and has been very useful in computer graphics.
- The Phong model is based on using the four vectors in the below figure.

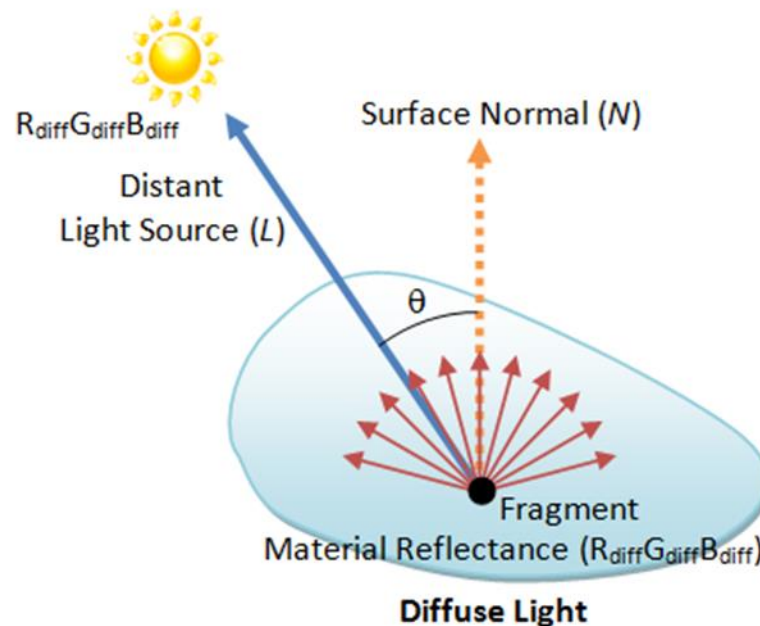


- The direction to the light source from P is given by the Vector L
- The viewer is located in the direction V from P
 - Either the viewer or the light source could be an infinite distance from the surface.
 - The direction of that reflected light leaves the surface depends on both the direction it came from and the orientation of the surface
- The local orientation at a point on the surface is given by normal vector N, which is the perpendicular to the surface at the point.
- If the surface is highly reflective, it will act like a mirror and most of the light will go off in the direction of a perfect reflector R.

- The Phong model considers four types of contributions to the shade at a point: diffuse reflection, specular reflection, ambient reflection and emissive light from the material.
- The Phong model uses RGB color and is applied independently to each primaries.
- Because the light model is additives, light from multiple sources can be added together.
- Thus the Phong model is computed once for each point source and the shade is determined by the sum of the contributions from

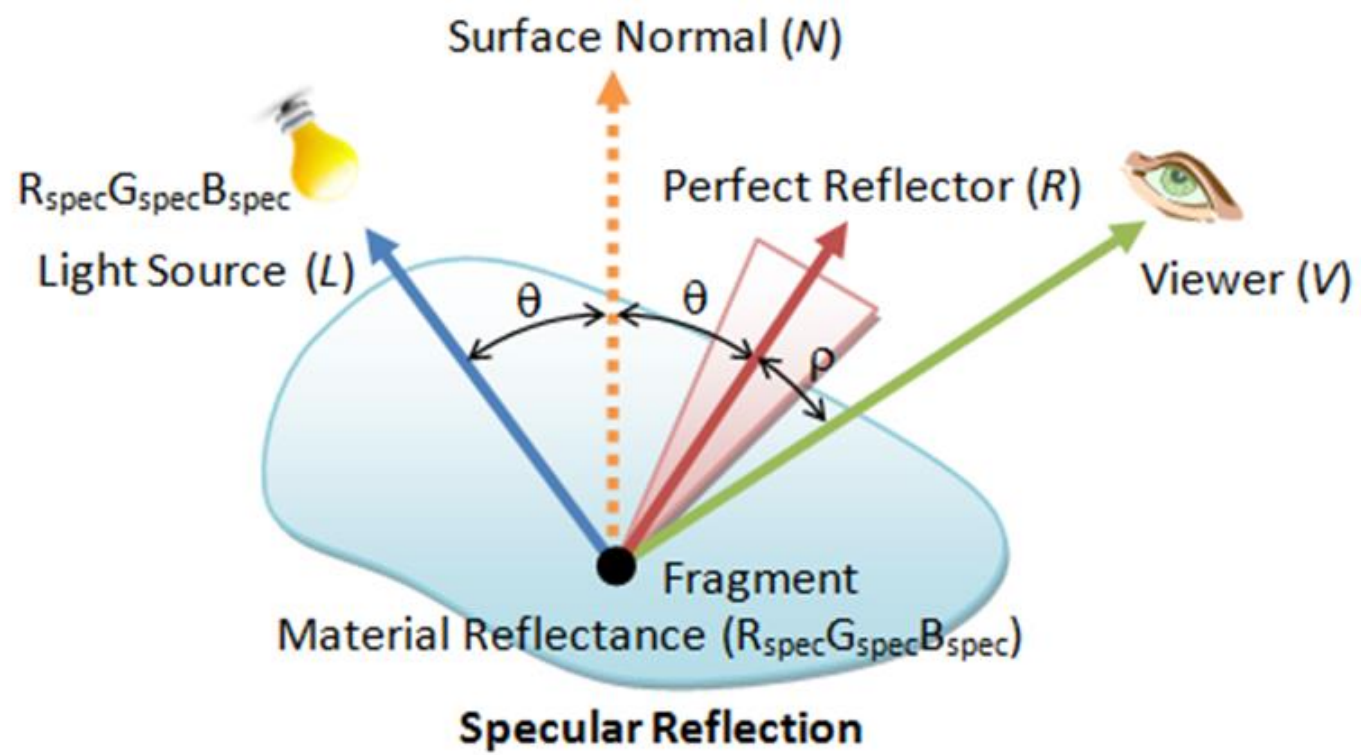
Diffuse Light

- Diffuse light models distant directional light source (such as the sun light). The reflected light is scattered equally in all directions, and appears the same to all viewers regardless of their positions, i.e., independent of viewer vector V . The strength of incident light depends on the angle between the light source L and the normal N , i.e., the dot product between L and N .



Specular Light

- The reflected light is concentrated along the direction of perfect reflector R . What a viewer sees depends on the angle (cosine) between V and R .
- The idea is that when light strikes a surface at some angle it is also reflected away at the same angle (on the other side of the normal). If the viewer is located exactly somewhere along the way of the reflected light ray it receives a larger amount of light than a viewer who is located further away.
- The end result of specular lighting is that objects will look brighter from certain angles and this brightness will diminish as you move away.



Ambient Light

- A constant amount of light applied to every point of the scene.
- Ambient light alone cannot communicate the complete representation of an object set in 3D space because all vertices are evenly lit by the same color and the object appears to be 2-dimensional

Emissive Light

- Some surfaces may emit light. With no additional light sources around, an object's color to which only emissive light component is applied has the same visual quality as an object with only ambient light applied to it.
- However, the mechanics of how any additional diffuse or specular light reacts with the surface of the same object with only emissive light applied to it is different.
- Resultant Color
 - The resultant color is the sum of the contribution in all the four components: (diffuse, Specular, ambient and emissive)

OpenGL's Lighting and Material

- OpenGL provides point sources (omni-directional), spotlights (directional with cone-shaped), and ambient light (a constant factor).
- Light source may be located at a fixed position or infinitely far away.
- Each source has separate ambient, diffuse, and specular components.
- Each source has RGB components. The lighting calculation is performed on each of the components independently (local illumination without considering the indirect lighting).

- In OpenGL, you need to enable the lighting state, and each of the light sources, identified via GL_LIGHT0 to GL_LIGHT n .
- glEnable(GL_LIGHTING); // Enable lighting
- glEnable(GL_LIGHT0); // Enable light source 0
- glEnable(GL_LIGHT1); // Enable light source 1

- Once lighting is enable, color assigned by glColor() are no longer used.
- Instead, the color depends on the light-material interaction and the viewer's position.
- You can use glLight() to define a light source (GL_LIGHT0 to GL_LIGHTn):
 - void glLight[if](GLenum lightSourceID, GLenum parameterName, type parameterValue);
 - void glLight[if]v(GLenum lightSourceID, GLenum parameterName, type *parameterValue);
 - // lightSourceID: ID for the light source, GL_LIGHT0 to GL_LIGHTn.
 - // parameterName: such as GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION.
 - // parameterValue: values of the parameter.

- For a point source, we can set its (x,y,z) location (GL_POSITION) and its diffuse, specular and ambient RGBA components.
- The default for GL_POSITION is (0, 0, 1) relative to camera coordinates, so it is behind the default camera position (0, 0, 0).
- GL_LIGHT0 is special, with default value of white (1, 1, 1) for the GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR components.
- You can enable GL_LIGHT0 right away by using its default settings. For other light IDs (GL_LIGHT1 to GL_LIGHTn), the default is black (0, 0, 0) for GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR.

Example

```
#include <GL/freeglut.h>
GLfloat dx=1.0,dy=1.0,dz=1.0;
GLfloat light_pos[]={1.0,2.0,3.0,1.0};
GLfloat theta=0.0;
```

```
void renderScene(){
    theta+=2.0;
    if(theta>360)
        theta-=360;

    glutPostRedisplay();
}
```

```
void init(){
    glEnable(GL_LIGHTING);
    glEnable (GL_LIGHT0);
}
```

```
void display(){
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_
    BIT);

    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(80,640/480,1,5);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(1,2,1,0,0,0,0,1,0);
    glPushMatrix();
    glRotatef(theta,dx,dy,dz);
    glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
    /*GLfloat qaAmbientLight[]={1,0,1,1.0};
    GLfloat qaDiffuseLight[]={0,1,1,1.0};
    GLfloat qaSpecularLight[]={1.0,1.0,1.0,1.0};
    glLightfv(GL_LIGHT0,GL_AMBIENT,qaAmbientLight);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,qaDiffuseLight);
    glLightfv(GL_LIGHT0,GL_SPECULAR,qaSpecularLight);
    glLightfv(GL_LIGHT0,GL_POSITION,light_pos);*/
    glPopMatrix();
    glutSolidCube(1);
    glutSwapBuffers();
}
```

```
int main(int argc, char** argv) {  
    glutInit(&argc, argv);  
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE |  
GLUT_RGBA);  
    glutInitWindowPosition(100,100);  
    glutInitWindowSize(500,500);  
    glutCreateWindow("Lighting2");  
    glutDisplayFunc(display);  
    glutIdleFunc(renderScene);  
    init();  
    glutMainLoop();  
    return 0;  
}
```

Material

- Similar to light source, a material has reflectivity parameters for specular (GL_SPECULAR), diffuse (GL_DIFFUSE) and ambient (GL_AMBIENT) components (for each of the RGBA color components), which specifies the fraction of light reflected.
- A surface may also emit light (GL_EMISSION).
- A surface has a shininess parameter (GL_SHININESS) - the higher the value, the more concentration of reflected-light in the small area around the perfect reflector and the surface appears to be shinier.
- Furthermore, a surface has two faces: front and back, that may have the same or different parameters.

- You can use `glMaterial()` function to specify these parameters for the front (`GL_FRONT`), back (`GL_BACK`), or both (`GL_FRONT_AND_BACK`) surfaces.
- The front face is determined by the surface normal (implicitly defined by the vertices with right-hand rule, or `glNormal()` function).
 - `void glMaterial[if](GLenum face, GLenum parameterName, type parameterValue)`
 - `void glMaterial[if]v(GLenum face, GLenum parameterName, type *parameterValues)`
 - `// face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK.`
 - `// parameterName: GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT, GL_AMBIENT_AND_DIFFUSE, GL_EMISSION, GL_SHININESS.`
- The default material has a gray surface (under white light), with a small amount of ambient reflection (0.2, 0.2, 0.2, 1.0), high diffuse reflection (0.8, 0.8, 0.8, 1.0), and no specular reflection (0.0, 0.0, 0.0, 1.0).

```

#include<iostream.h>
#include <GL/glut.h>
void display();
double angle=0;
double i=0.3;
void idle() {
glutPostRedisplay();
}
void init()
{
    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        GLfloat qaAmbientLight[]={0,0,1,1.0};
        GLfloat qaDiffuseLight[]={0,0,1,1.0};
        GLfloat qaSpecularLight[]={1.0,1.0,1.0,1.0};
        glLightfv(GL_LIGHT0,GL_AMBIENT,qaAmbientLight);
        glLightfv(GL_LIGHT0,GL_DIFFUSE,qaDiffuseLight);
        glLightfv(GL_LIGHT0,GL_SPECULAR,qaSpecularLight);
        GLfloat qaLightPosition[]={2,0,0,1.0};
        glLightfv(GL_LIGHT0,GL_POSITION,qaLightPosition);
}

```

```

int main(int argc, char* argv[]){
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
GLUT_DEPTH);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Animation");
    init();
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    glutMainLoop();
    return 0;
}

```

```
void display(){
```

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_B  
IT);
```

```
GLfloat qaBlue[]={0.0,0.0,1.0,1.0};
```

```
GLfloat qaGreen[]={0.0,1.0,0.0,1.0};
```

```
GLfloat qaWhite[]={1.0,1.0,1.0,1.0};
```

```
GLfloat qaRed[]={1,0,0,1};
```

```
GLfloat qaYellow[]={1,1,0,1};
```

```
glMaterialfv(GL_FRONT,GL_AMBIENT,qaRed);
```

```
glMaterialfv(GL_FRONT,GL_DIFFUSE,qaBlue);
```

```
glMaterialfv(GL_FRONT,GL_SPECULAR,qaWhite);
```

```
glMaterialf(GL_FRONT,GL_SHININESS,100.0);
```

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

```
gluPerspective(60,640/480,1,15);
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

```
gluLookAt(0, 0, 2, 0, 0, 0, 1, 1, 0);
```

```
glRotatef(angle, 1, 0.0, 0 );
```

```
glRotatef(angle, 0.0, 1.0, 0.0 );
```

```
glRotatef(angle, 0.0, 0.0, 1.0 );
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glNormal3f(0.0,0.0,1.0);
```

```
glBegin(GL_POLYGON);//the bottom
```

```
glVertex3f(0.5, 0,0 );
```

```
glVertex3f(0.5, 0.0, -0.5 );
```

```
glVertex3f(0,0,-0.5 );
```

```
glVertex3f(0,0,0 );
```

```
glEnd();
```

```
glBegin(GL_POLYGON);//right side
```

```
glVertex3f(0.5, 0,0 );
```

```
glVertex3f(0.5, 0.0, -0.5 );
```

```
glVertex3f(0.5, 0.8, -0.5 );
```

```
glEnd();
```

```
glBegin(GL_TRIANGLES); //back side
glVertex3f(0.5, 0.0, -0.5 );
glVertex3f(0,0,-0.5 );
glVertex3f(0.5, 0.8, -0.5 );
glEnd();
glBegin(GL_TRIANGLES); //left side
```

```
glVertex3f(0,0,-0.5 );
glVertex3f(0,0,0 );
glVertex3f(0.5, 0.8, -0.5 );
glEnd();
glBegin(GL_TRIANGLES); //front
glVertex3f(0,0,0 );
glVertex3f(0.5, 0,0 );
glVertex3f(0.5, 0.8, -0.5 );
glEnd();
angle+=0.01;
glutSwapBuffers();
}
```