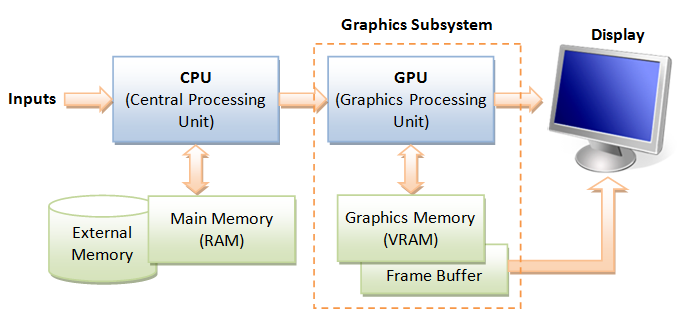
**OpenGL (Open Graphics Library)**

OpenGL (Open Graphics Library) is the computer industry's standard application program interface ( [API](https://searchmicroservices.techtarget.com/definition/application-program-interface-API) ) for defining 2-D and 3-D graphic images. Prior to OpenGL, any company developing a graphical application typically had to rewrite the graphics part of it for each [operating system](https://whatis.techtarget.com/definition/operating-system-OS) [platform](https://searchservervirtualization.techtarget.com/definition/platform) and had to be cognizant of the graphics hardware as well. With OpenGL, an application can create the same effects in any operating system using any OpenGL-adhering graphics adapter.

OpenGL specifies a set of "commands" or immediately executed functions. Each command directs a drawing action or causes special effects. A list of these commands can be created for repetitive effects. OpenGL is independent of the windowing characteristics of each operating system, but provides special "glue" routines for each operating system that enable OpenGL to work in that system's windowing environment. OpenGL comes with a large number of built-in capabilities requestable through the API. These include hidden surface removal, alpha blending (transparency), [antialiasing](https://whatis.techtarget.com/definition/antialiasing) , texture mapping, [pixel](https://whatis.techtarget.com/definition/pixel) operations, viewing and modeling transformations, and atmospheric effects (fog, smoke, and haze).

Silicon Graphics, makers of advanced graphics [workstation](https://searchmobilecomputing.techtarget.com/definition/workstation) s, initiated the development of OpenGL. Other companies on the industry-wide Architecture Review Board include DEC, Intel, IBM, Microsoft, and Sun Microsystems. There is no cost (other than learning) to developing an application using the OpenGL API. Microsoft offers free downloads of the OpenGL libraries for its Windows systems. Although OpenGL is not itself a development "toolkit," such toolkits are available, including Silicon Graphics [object-oriented programming](https://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP) 3D graphics toolkit, Open Inventor



Graphics with Open GL

Object and image space methods

In a 3D objects and viewing specification, we wish to determine which lines or surfaces of the objects are visible, so that we can display only the visible lines or surfaces. This process is known as hidden surfaces or hidden line elimination, or visible surface determination. The hidden line or hidden surface algorithm determines the lines, edges, surfaces or volumes that are visible or invisible to an observer located at a specific point in space. These algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. These two approaches are called **object-space methods** or object precision methods and **image-space methods**, respectively. When we view a picture containing non-transparent objects and surfaces, then we cannot see those objects from view which are behind from objects closer to eye. We must remove these hidden surfaces to get a realistic screen image. The identification and removal of these surfaces is called **Hidden-surface problem**.

There are two approaches for removing hidden surface problems − **Object-Space method and Image-space method.**

**Object-space method:-** Object-space method is implemented in the physical coordinate system in which objects are described. It compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. Object-space methods are generally used in line-display algorithms.

**Image-Space method:-** Image space method is implemented in the screen coordinate system in which the objects are viewed. In an image-space algorithm, visibility is decided point by point at each pixel Position on the view plane. Most hidden line/surface algorithms use image-space methods.

Hidden line and surface removal techniques (Object and image space method)

When we view a picture containing non-transparent objects and surfaces, then we cannot see those objects from view which are behind from objects closer to eye. We must remove these hidden surfaces to get a realistic screen image. The identification and removal of these surfaces is called **Hidden-surface problem**.

There are two approaches for removing hidden surface problems − **Object-Space method** and **Image-space method**. The Object-space method is implemented in physical coordinate system and image-space method is implemented in screen coordinate system.

When we want to display a 3D object on a 2D screen, we need to identify those parts of a screen that are visible from a chosen viewing position.

## Depth Buffer (Z-Buffer) Method

This method is developed by Cutmull. It is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest (visible) surface.

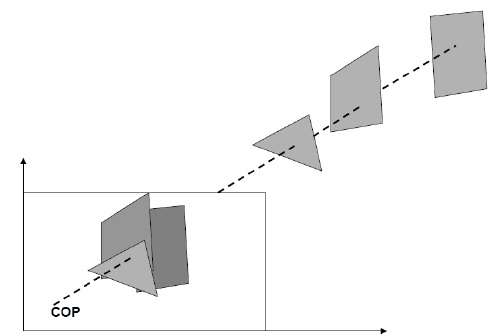
In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest (smallest z) surface determines the color to be displayed in the frame buffer.

It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named **frame buffer** and **depth buffer,** are used.

**Depth buffer** is used to store depth values for (x, y) position, as surfaces are processed (0 ≤ depth ≤ 1).

The **frame buffer** is used to store the intensity value of color value at each position (x, y).

The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates front clipping pane.



### Algorithm

**Step-1** − Set the buffer values −

Depthbuffer (x, y) = 0

Framebuffer (x, y) = background color

**Step-2** − Process each polygon (One at a time)

For each projected (x, y) pixel position of a polygon, calculate depth z.

If Z > depthbuffer (x, y)

Compute surface color,

set depthbuffer (x, y) = z,

framebuffer (x, y) = surfacecolor (x, y)

### Advantages

* It is easy to implement.
* It reduces the speed problem if implemented in hardware.
* It processes one object at a time.

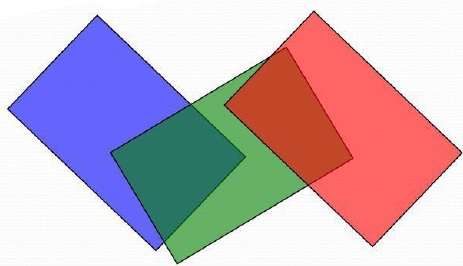
### Disadvantages

* It requires large memory.
* It is time consuming process.

## A-Buffer Method

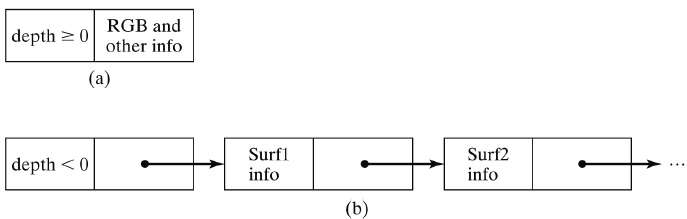
The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw (REYES).

The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer.



Each position in the A-buffer has two fields −

* **Depth field** − It stores a positive or negative real number
* **Intensity field** − It stores surface-intensity information or a pointer value



If depth >= 0, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.

If depth < 0, it indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. The surface buffer in the A-buffer includes −

* RGB intensity components
* Opacity Parameter
* Depth
* Percent of area coverage
* Surface identifier

The algorithm proceeds just like the depth buffer algorithm. The depth and opacity values are used to determine the final color of a pixel.

## Methods of generating non planar surfaces

## Bezier Curves

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

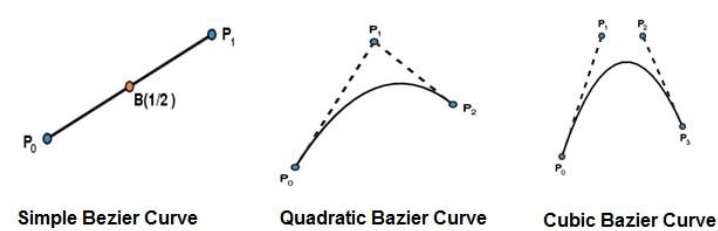
∑k=0nPiBni(t)∑k=0nPiBin(t)

Where pipi is the set of points and Bni(t)Bin(t) represents the Bernstein polynomials which are given by −

Bni(t)=(ni)(1−t)n−itiBin(t)=(ni)(1−t)n−iti

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

The simplest Bézier curve is the straight line from the point P0P0 to P1P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



## Properties of Bezier Curves

Bezier curves have the following properties −

* They generally follow the shape of the control polygon, which consists of the segments joining the control points.
* They always pass through the first and last control points.
* They are contained in the convex hull of their defining control points.
* The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
* A Bezier curve generally follows the shape of the defining polygon.
* The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
* The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
* No straight line intersects a Bezier curve more times than it intersects its control polygon.
* They are invariant under an affine transformation.
* Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
* A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

## B-Spline Curves

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

* First, the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve.
* The second limiting characteristic is that the value of the blending function is nonzero for all parameter values over the entire curve.

The B-spline basis contains the Bernstein basis as the special case. The B-spline basis is non-global.

A B-spline curve is defined as a linear combination of control points Pi and B-spline basis function Ni,Ni, k (t) given by

C(t)=∑ni=0PiNi,k(t),C(t)=∑i=0nPiNi,k(t), n≥k−1,n≥k−1, tϵ[tk−1,tn+1]tϵ[tk−1,tn+1]

Where,

* {pipi: i=0, 1, 2….n} are the control points
* k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree k - 1,
* the Ni,k(t)Ni,k(t) are the “normalized B-spline blending functions”. They are described by the order k and by a non-decreasing sequence of real numbers normally called the “knot sequence”.

ti:i=0,...n+Kti:i=0,...n+K

The Ni, k functions are described as follows −

Ni,1(t)={1,0,ifuϵ[ti,ti+1)OtherwiseNi,1(t)={1,ifuϵ[ti,ti+1)0,Otherwise

and if k > 1,

Ni,k(t)=t−titi+k−1Ni,k−1(t)+ti+k−tti+k−ti+1Ni+1,k−1(t)Ni,k(t)=t−titi+k−1Ni,k−1(t)+ti+k−tti+k−ti+1Ni+1,k−1(t)

and

tϵ[tk−1,tn+1)tϵ[tk−1,tn+1)

## Properties of B-spline Curve

B-spline curves have the following properties −

* The sum of the B-spline basis functions for any parameter value is 1.
* Each basis function is positive or zero for all parameter values.
* Each basis function has precisely one maximum value, except for k=1.
* The maximum order of the curve is equal to the number of vertices of defining polygon.
* The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
* B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
* The curve exhibits the variation diminishing property.
* The curve generally follows the shape of defining polygon.
* Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
* The curve line within the convex hull of its defining polygon

**Gouraud and phong shading models**

**Gouraud Shading:**

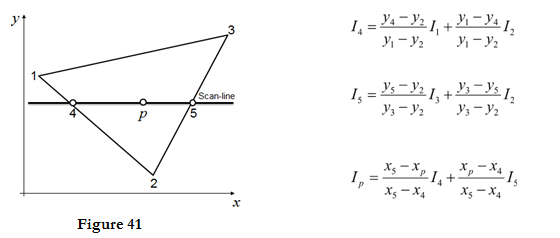
1. Gouraud surface shading was developed in the 1970s by Henri Gouraud.
2. It is the interpolation technique.
3. Intensity levels are calculated at each vertex and interpolated across the surface.
4. Intensity values for each polygon are matched with the values of adjacent polygons along the common edges.
5. This eliminates the intensity discontinuities that can occur in flat shading.
6. To render a polygon, Gouraud surface rendering proceeds as follows:
   * Determine the average unit normal vector at each vertex of the polygon.
   * Apply an illumination model at each polygon vertex to obtain the light intensity at that position.
   * Linearly interpolate the vertex intensities over the projected area of the polygon
7. The average unit normal vector at V is given as:

Nv=N1+N2+N3+N4|N1+N2+N3+N4|Nv=N1+N2+N3+N4|N1+N2+N3+N4|

1. More generally as

Nv=Σni=1NiΣni=1NiNv=Σi=1nNiΣi=1nNi

1. Illumination values are linearly interpolated across each scan-line as shown in figure 41.



1. The intensities at point 4 can be interpolated from intensities 1 and 2.
2. Similarly, the intensities at point 5 can be interpolated from intensities 2 and 3.
3. Therefore the intensities of interaction points 4 and 5 are calculated from scan line.

**Advantages:**

i. It removes the intensity discontinuity which exists in constant shading model.

ii. It can be combined with hidden surface algorithm to fill in the visible polygons along each scan line.

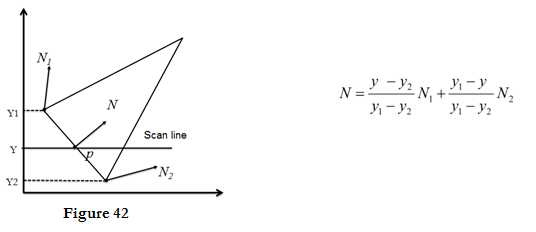
**Disadvantages:**

i. Gouraud shading has a problem with specular reflections.

ii. Gouraud shading can introduce anomalies known as Mach bands.

**Phong Shading:**

1. A more accurate interpolation based approach for rendering a polygon was developed by Phong Bui Tuong.
2. Basically the Phong surface rendering model is also called as normal-vector interpolation rendering.
3. It interpolates normal vectors instead of intensity values.
4. To render a polygon, Phong surface rendering proceeds as follows:
5. Determine the average unit normal vector at each vertex of the polygon.
6. Linearly interpolate the vertex normal over the projected area of the polygon.
7. Apply an illumination model at positions along scan lines to calculate pixel intensities using the interpolated normal vectors as shown in figure 42



**Advantages:**

i. It displays more realistic highlights on a surface.

ii. It greatly reduces the Mach band effect.

iii. It gives more accurate results.

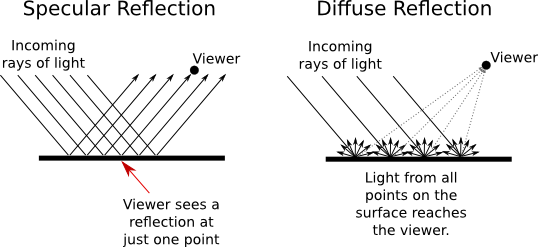
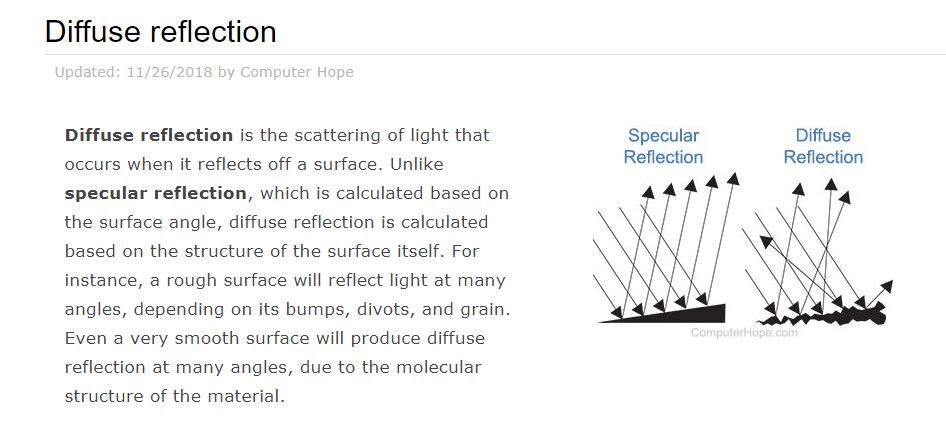
**Disadvantages:**

It requires more calculations and greatly increases the cost of shading steeply.

What is ambient light ?

A surface that is not exposed directly to a light source still will be visible if nearby objects are illuminated. In our basic illumination model, we can set a general level of brightness for a scene. This is a simple way to model the combination of light reflections from various surfaces to produce a uniform illumination called the ambient light, or background light.

Ambient light has no spatial or directional Characteristics. The amount of ambient light incident on each object is a constant for all surfaces and over all directions.



## Introduction to Diffuse Reflection

Every time I go to the beach, I am fascinated by all of the colored pebbles I find. Tiny stones of pink, yellow, blue, and violet flash under the shallow waves. I can't resist the urge to collect them, and I usually go home with a pocketful of my favorite pebbles. But once I get them washed and dried, I am often disappointed with how dull they all appear. Where did their vibrant colors go? What happened to their glossy sheens? I should have known from the beginning that the dry pebbles would look different because of the effects of **diffuse reflection**.

## Specular vs. Diffuse Reflection

We've already learned that the reflection of waves involves a change in the direction of waves when they strike a surface. When talking about reflection, we think of waves as straight-line rays. The incident ray is the ray that initially strikes a surface. The reflected ray, obviously, is the one that reflects off a surface. The law of reflection tells us that the angle of reflection is equal to the angle of incidence. In other words, the angle at which the incident ray strikes the surface is going to be the same as the angle at which it reflects. Rays and angles are convenient ways to talk about single waves, but sources of sound and light typically send out many waves at once. How do we talk about the law of reflection when more than one wave is involved?

All we have to do is imagine a whole bunch of rays parallel to one another. This picture below could represent a concentrated beam of light waves, say, from a flashlight. Since each ray is hitting the surface at the same angle - the angle of incidence - then each ray reflects at the same angle - the angle of reflection. A person standing in the path of the reflected rays would see a perfect image of the flashlight's beam. It wouldn't be much different from staring into the flashlight itself! This type of reflection, in which a smooth surface causes reflected rays to travel in the same direction, is called **specular reflection**.

|  |
| --- |
| Specular Reflection Rays Parallel |
| ***When the light rays hit a smooth surface, the rays are parallel to each other.*** |

Specular reflection is best known as the type of reflection you get from a mirror. It also occurs off of other smooth surfaces like glossy tabletops, car windows, and very still water. Specular reflection from a calm lake occurs because the surface is so flat that all the reflected rays bounce off in the same direction.

But, if a gust of wind disturbed the water, the resulting ripples would cause a different phenomenon called **diffuse reflection**. Diffuse reflection occurs when a rough surface causes reflected rays to travel in different directions. Most everyday objects exhibit diffuse reflection because of the tiny imperfections on the surface of the material. A piece of paper may look smooth on the surface, but at the microscopic level, the tiny fibers make it rough. Upholstery and clothing exhibit diffuse reflection because of the minute roughness of the fabric. Even a leaf has an element of roughness to it because of the multidimensional nature of the cells on the surface.

In diffuse reflection, each individual ray strikes a part of the surface that is oriented in a different direction. The law of reflection still applies, but the normal is different for each ray. So, the reflected rays end up going in all directions. As you can see below, one incident ray reflects over in one direction, another incident ray reflects elsewhere, and another one reflects somewhere else. The effect of all these rays going everywhere is that all of the waves are spread out. Diffuse reflection is the reason why you don't see your image reflected in most everyday objects.

|  |
| --- |
| Diffuse Reflection Ray Diagram |
| ***How reflective rays travel during diffuse reflection*** |