Computer Graphics and GPGPU Programming

Donato D'Ambrosio

Department of Mathematics and Computer Science
and Center of Excellence for High Performace Computing
Cubo 22B, University of Calabria, Rende 87036, Italy
mailto: donato.dambrosio@unical.it
homepage: http://www.mat.unical.it/~donato

Academic Year 2018/19

Table of contents

Algorithms

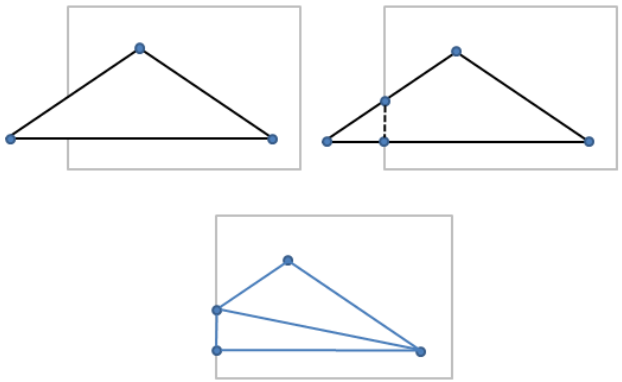
Algorithms

Stages of the OpenGL's Graphics Process

- The graphics process can be split in the following stages:
  - Modeling (the 3D scene is conceptually defined by means of vertices in a vectorial context)
  - Vertex Processing
    - Primitives (e.g. triangles) are defined
    - Primitives laying outside the clip space are discarded (clipping)
    - Hidden primitives are discarded (hidden surface removal)
  - Rasterization
    - The scene is converted into a raster image (image's elements are called fragments)
  - Fragment processing (a color is assigned to each fragment)
  - Output merging (combine the fragments of all primitives into color pixels for the display)

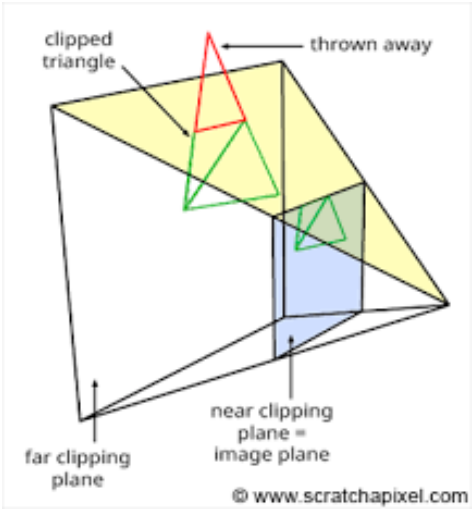
Clipping

- Clipping is the process that discards (entirely or partially) the primitives that are outside the clip space
- The primitives within the clip space are accepted, while the other are eliminated or rejected
- The primitives that are partially within the clip space are cut and new vertices added in order to obtain a new primitive that is entirely within the clip space



Clipping

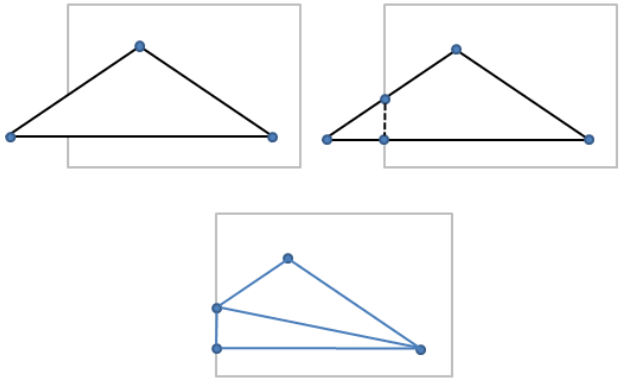
- Evaluating the intersections with the clip planes is the most important computational effort in clipping algorithm



- Accordingly, it is important to minimize the number of intersections (since they require floating point operations)

Cohen-Sutherland's clipping algorithm

- Let consider the two-dimensional case (it is conceptually equivalent to the three-dimensional one)



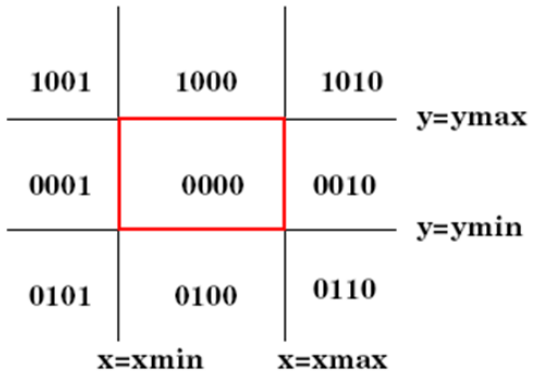
- The edges of the clipping plane are ideally extended to the infinitive by forming 9 regions
- A 4 bit code $O = b_0 b_1 b_2 b_3$, called outcode, is assigned to each region

Cohen-Sutherland's clipping algorithm

Outcodes are defined by the following rule:

$$b_0 = \begin{cases} 1 & \text{if } y > y_{max} \\ 0 & \text{if } y \leq y_{max} \end{cases}$$

- Similarly, $b_1 = 1$ if $y < y_{min}$
- b_2 and b_3 are defined by relations between x and the left and right edges of the clip region
- Note that the clip region has outcode $O = 0000$

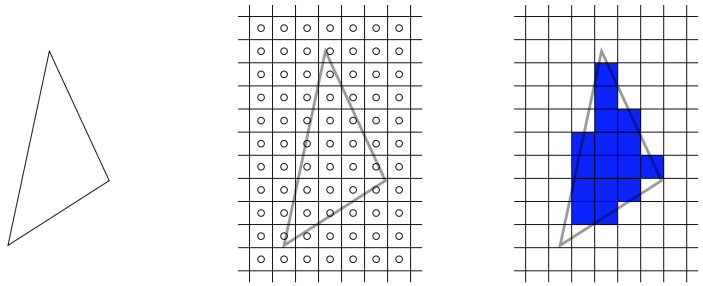


Cohen-Sutherland's clipping algorithm

- Case 1: $O_1 = O_2 = 0000$
  Both outcode are 0000. The segment is INTERNAL
- Case 2: $O_1 \& O_2 \neq 0000$
  The extremes are both up, down, to the left or to the right of the clip window. The segment is EXTERNAL
- Case 3: $O_1 \& O_2 = 0000$
  The extremes are external but the line can not be rejected since it could intersect the window. Intersections with the clip window's edges must be computed and if the line actually intersects the clip region new vertces evaluated
- Case 4: $O_1 \neq 0000$; $O_2 = 0000$, or *viceversa*
  The line is partially outside the clip window. In this case, the vertex outside the window is replaced by a ne vertex resulting from the intersection between the line and the clip window's edge

Scan conversion of segments

- Starting from the vertices of the primitive projected into the two-dimensional plane, the rasterization or scan conversion builds the primitive as a set of fragments

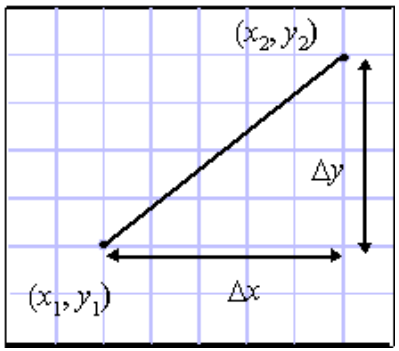


- Let us suppose the vertices composing the geometric primitives have already been projected on a $n \times m$ matrix with the origin located at the bottom-left pixel
- Let us suppose that each pixel is a square with the center at the pixel coordinates and side equal to the distance between two adjacent pixels

The DDA line algorithm

- Let us suppose to have a segment defined by its extremes (x_1, y_1) and (x_2, y_2)
- The slope of the segment is defined as:

$$m = \Delta y / \Delta x$$



- The most simple scan conversion algorithm evaluates m, increments x (starting from the left extreme), and computes $y_i = mx_i + h$, where h is the ordinata of the intersection between the y axis and the line, for each x_i

The DDA line algorithm

- The above strategy is inefficient since each iteration requires one floating point multiplication and one floating point addition
- It is possible to avoid the multiplication by adopting an *incremental strategy* so that one pixel can be computed based on the previous one
- This algorithm is called DDA[1]
- Each time x is incremented of Δx, the corresponding variation of y must be $\Delta y = m\Delta x$
- Ranging form x_1 to x_2, x is incremented by 1 (i.e. the distance between two pixels) and then y is incremented by $\Delta y = m$

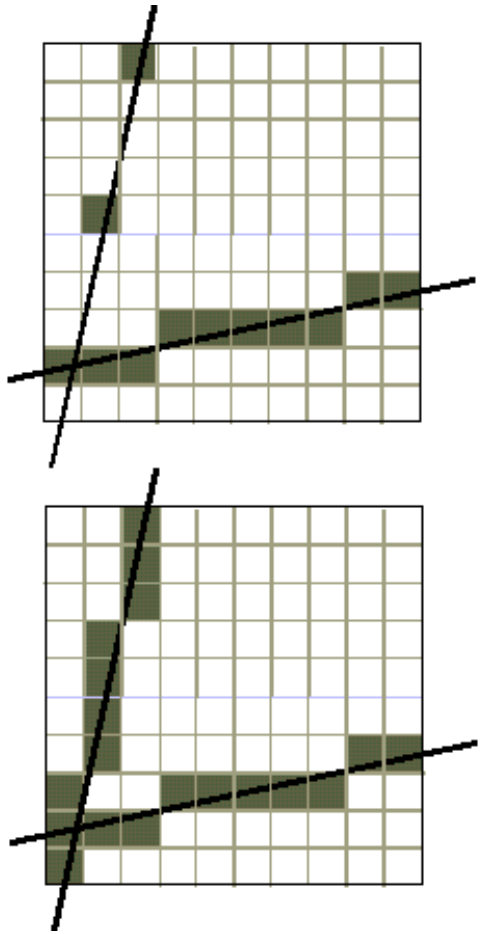
[1] The name comes from the Digital Differential Analyzer, a mechanical device able to solve differential equations by applying a numerical methods

The DDA line algorithm

- Let us assume that $0 \leq m \leq 1$ (the other values can be treated equivalently)

```
int x;
float dy, dx, y, m;

dy = y2 - y1;
dx = x2 - x1;
m = dy/ dx;
y = y1;
for (x = x1, x <= x2, x++)
{
   writePixel(x, round(y),
       line_color);
   y += m;
}
```



Bresenham's line algorithm

- The DDA line algorithm is straightforward and easy to implement but execs a floating point operation for each line's pixel
- At the contrary, the Bresenham's line algorithm only performs integer operations
- Bresenham's algorithm is faster than DDA and, for this reason, it has become the most used line algorithm in Computer Graphics

***

- Let us suppose to have a segment defined by its extremes (x_1, y_1) and (x_2, y_2)

Bresenham's line algorithm

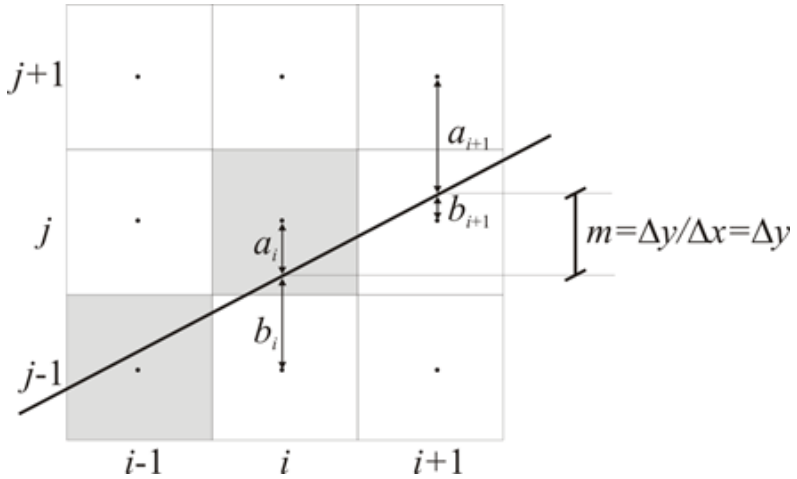
- Let us suppose we are at an intermediate step of the algorithm and the pixel (i, j) was the last to be switched on
- As a consequence, when $x = i$ the line $y = mx + h$ intersects the pixel (i, j)
- When $x = i + 1$, the condition $0 \leq m \leq 1$ implies that only one of the pixels $(i + 1, j)$ and $(i + 1, j + 1)$ can be switched on
- The choice can be expressed by means of a decision variable

$$d = b - a$$

where a and b are the measures of lines starting from the centers of pixels $(i + 1, j + 1)$ and $(i + 1, j)$, respectively (as shown in the next figure), to the line $y = mx + h$ and parallel to the y axis

Bresenham's line algorithm

- Definition of the decision variable $d = b - a$

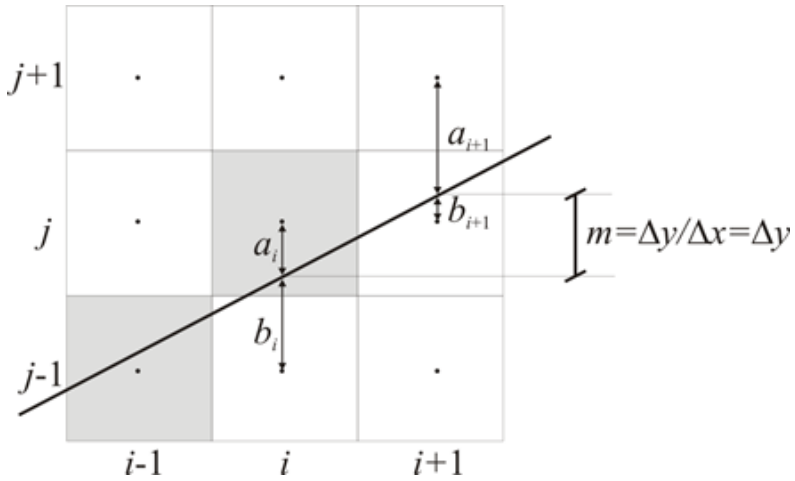


- If $d > 0$ then the line $y = mx + h$ is closest to the pixel $(i + 1, j + 1)$
- If $d \leq 0$ then the line $y = mx + h$ is closest to the pixel $(i + 1, j)$

- d can be computed without the need of floating point operations by means of a recurrence formula where d_{i+1} is a function of d_i

Bresenham's line algorithm

Case $d_i > 0$



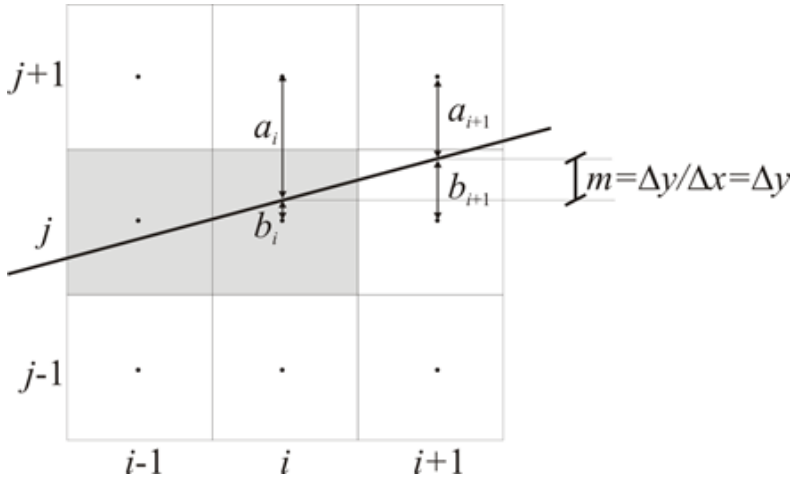
$$1 + b_{i+1} = b_i + m \Rightarrow b_{i+1} = b_i + m - 1$$

$$a_{i+1} = 1 - b_{i+1} = 1 - (b_i + m - 1) = 1 - (1 - a_i + m - 1) = a_i - m + 1$$

$$d_{i+1} = b_{i+1} - a_{i+1} = b_i - a_i + 2m - 2 = d_i + 2m - 2 = d_i + 2(m - 1)$$

Bresenham's line algorithm

Case $d_i \leq 0$



$$b_{i+1} = b_i + m$$

$$a_{i+1} = a_i - m$$

$$d_{i+1} = b_{i+1} - a_{i+1} = b_i - a_i + 2m = d_i + 2m$$

Bresenham's line algorithm

- So, we have

$$d_{i+1} = \begin{cases} d_i + 2(m-1) & \text{if } d > 0 \\ d_i + 2m & \text{if } d \leq 0 \end{cases}$$

- Since we are interested in the sign of d, we can multiply by $\Delta x = x_2 - x_1$ (that is a positive quantity)

$$d_{i+1}\Delta x = \begin{cases} d_i\Delta x + 2(m-1)\Delta x = d_i\Delta x + 2(\Delta y - \Delta x) & \text{if } d > 0 \\ d_i\Delta x + 2m\Delta x = d_i\Delta x + 2\Delta y & \text{if } d \leq 0 \end{cases}$$

- Eventually, we can impose $d_i = d_i\Delta x$ (since both of them have the same sign)

$$d_{i+1} = d_i + \begin{cases} 2(\Delta y - \Delta x) & \text{if } d > 0 \\ 2\Delta y & \text{if } d \leq 0 \end{cases}$$

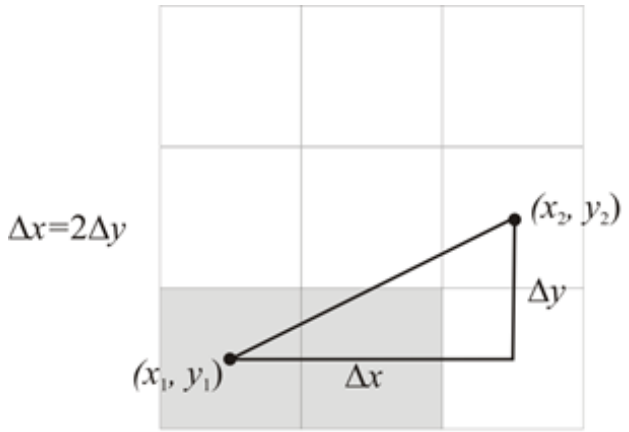
Bresenham's line algorithm

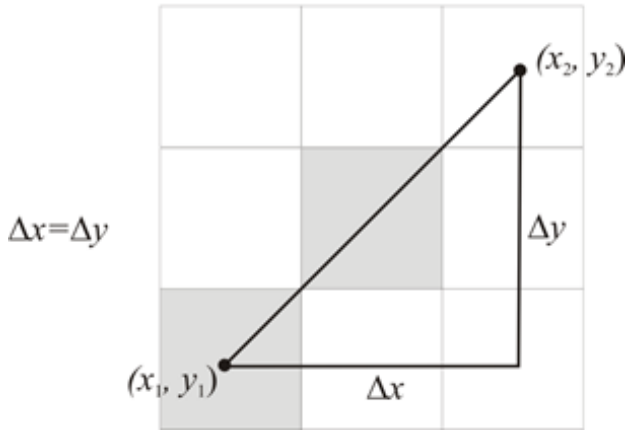
As regard the initial value, d can be set as follows:

$$d = 2(y_2 - y_1) - (x_2 - x_1)$$

In this manner

- $d > 0$ when the angle is > 22.5 degs (and the North-East pixel is switched on)
- $d \leq 0$ when the angle is ≤ 22.5 degs (and the East pixel si switched on)



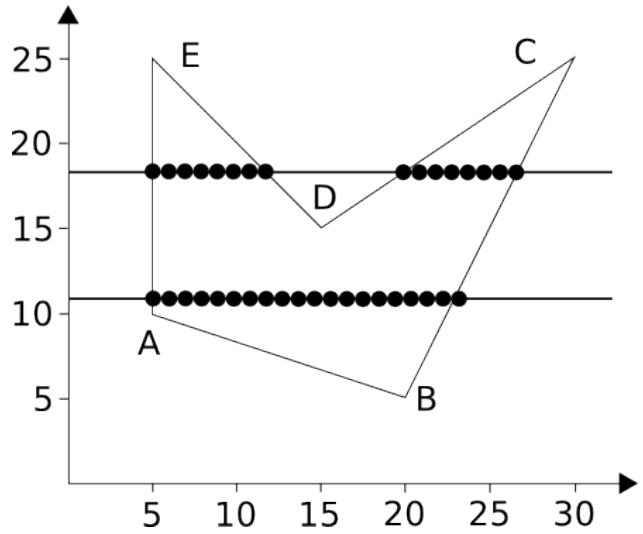


Bresenham's line algorithm

```
dx = x2 - x1; dy = y2 - y1;
d = dy * 2 - dx;
incrE = 2 * dy; incrNE = 2 * (dy - dx);
x = x1; y = y1; writePixel(x, y, value);
while (x < x2) {
  if (d <= 0) {
    d += incrE;
    x++;
  }
  else
  {
    d += incrNE;
    x++;
    y++;
  }
  writePixel(x, y, value);
}
```

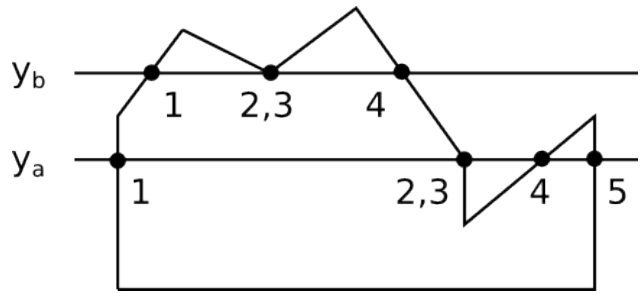
Scanline polygon algorithm

- The scanline algorithm is the reference algorithm for the scan conversion of polygons
- The algorithm uses scanlines to detect which pixels belong to the polygon



Scanline polygon algorithm

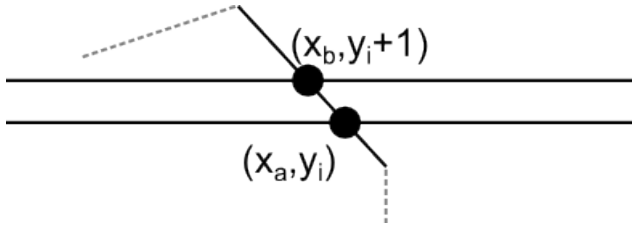
- For each scanline, the intesections with the polygon's edges are evaluated and labeled with integer numbers (starting from 1)
- The pixels between odd-even couples (1-2, 3-4, ecc.) are switched on



- Note that the scanline y_a intersects the edges 5 times: In such a case, the intersections 2 and 3 are considered as a single intersection since they are obtained on monotone edges
- At the contrary, the intersections 2 and 3 of the scanline y_b are valid since the common vertex represent a local minimum (or maximum)

Scanline polygon algorithm

- The rasterization process starts from the bottom and from left to right
- Acoordingly, each next scanline differs by 1 along y



- The edge's slope is thus

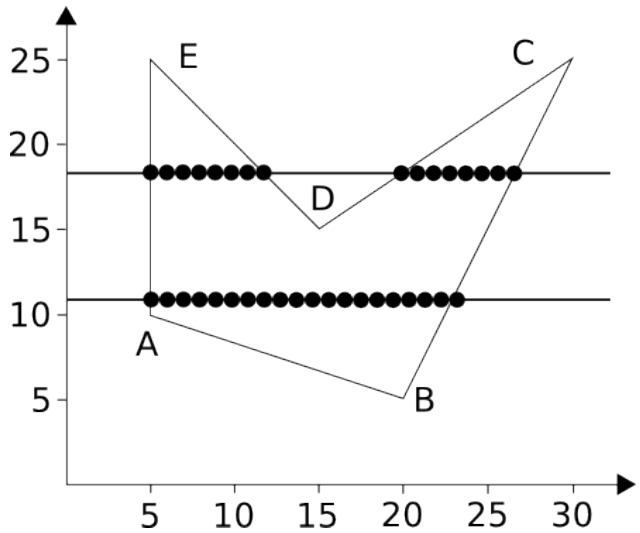
$$m = \frac{y_{i+1} - y_i}{x_b - x_a} = \frac{1}{x_b - x_a}$$

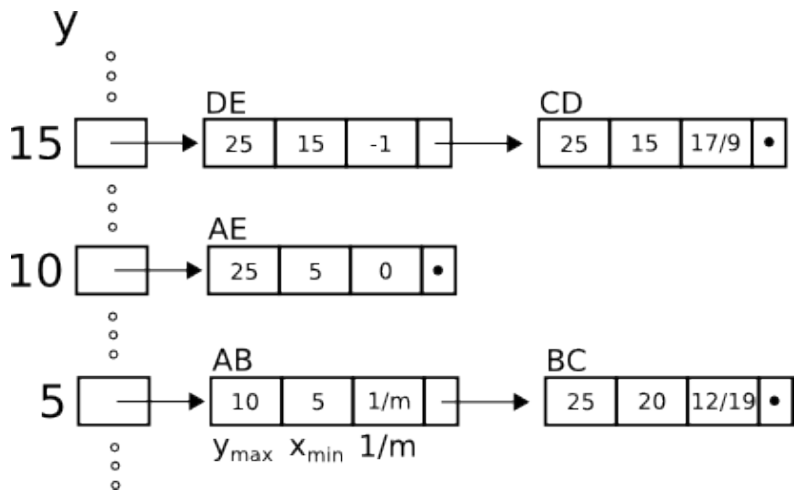
and therefore

$$x_b = x_a + \frac{1}{m}$$

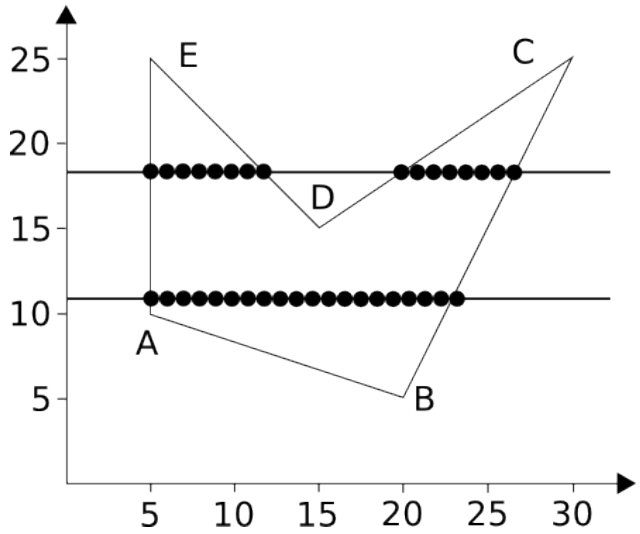
Scanline polygon algorithm

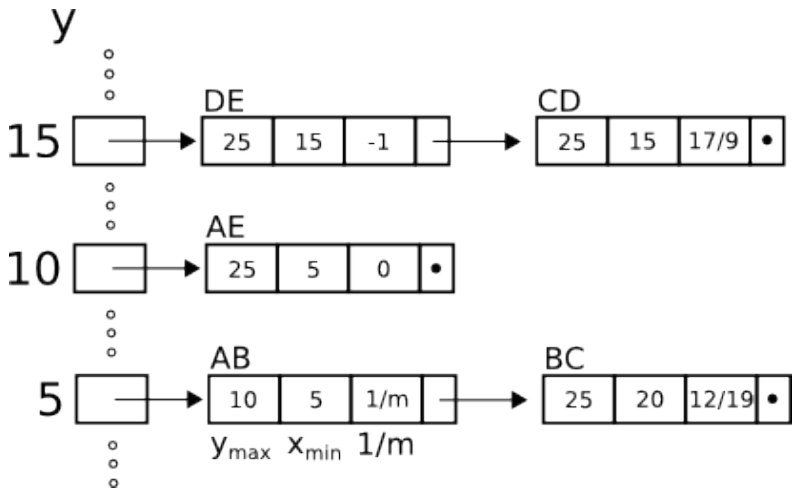
- As a consequence, after the firt intesection (x_a, y_a), the subsequent ones can be evaluated as $(x_b, y_b) = (x_a + 1/m, y_a + 1)$
- The algorithm uses an edge table (ET), which is an array of pointers with as many entries as the framebuffer's scanlines





Scanline polygon algorithm

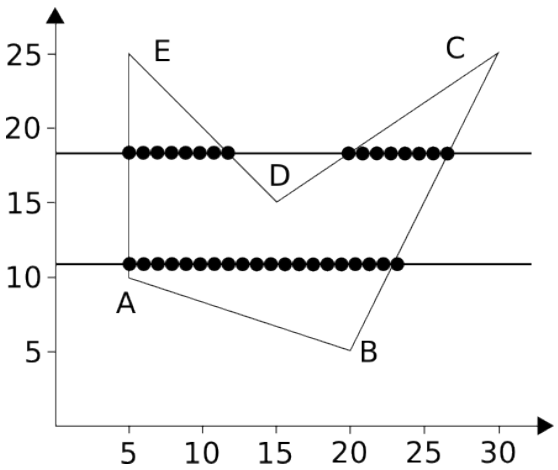


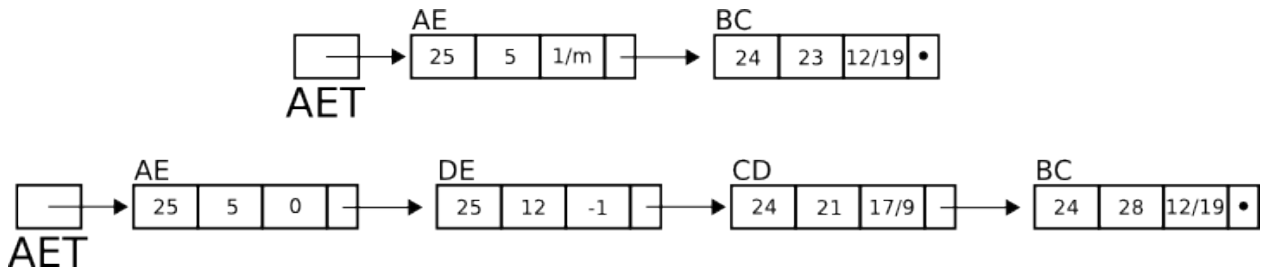


- If non-null, the j^{th} ET entry points to the list of edges that have minimum y equal to j
- The edges of each ET entry are ordered per minimum abscissa (x_{min}) in increasing order
- The inverse of the edge's slope ($1/m$) is also stored

Scanline polygon algorithm

- An active edge table (AET) is also considered, which stores information about the edges intersected by the current scanline
- Differently from ET data, the second edge's information stores the current abscissa (of the scanline-edge intesection), inspite of the minimun one
- The figure below shows the AET content at two different stages of the algorithm, corresponding to the scanlines $y = 11$ and $y = 18$





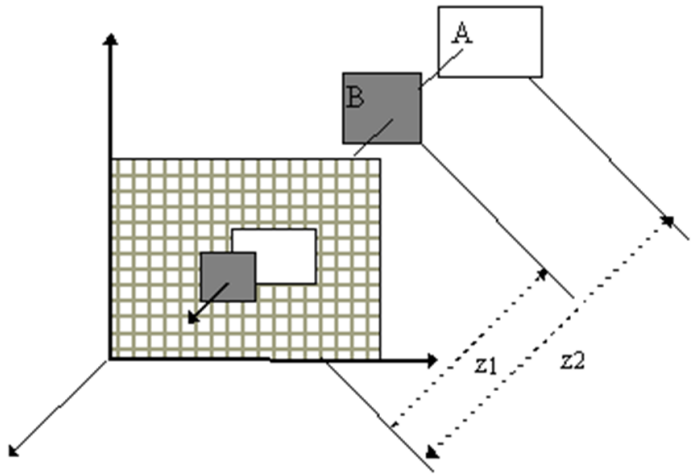
Scanline polygon algorithm

Once the ET has been defined, the algorithm processes the polygon as follows:

1. set the ordinate of the current scanline to the ordinate of the firt edge of the first entry of the ET ($y = 5$ in the example);
2. initialize AET to null (empty)
3. repeat the following steps until ET and AET are non empty:
   - move the edges with ordinate y to the AET
   - swhitch on the pixels between the odd-even intersections (1-2, 3-4, etc)
   - remove the edges for which $y = y_{max}$ from AET (since they will not be interesected by the next scanline)
   - increments y by 1 and x by $1/m$ (this latter only if the edge is not vertical)

Hidden surface removal

- OpenGL rasterizes the primitives (e.g. triangles) in the order they appear into the vertex buffer object (VBO)
- In the case an object that is far from the observer is rasterized after another object that is nearer, the former can be erroneously overwriten (since it was first rasterized)
- An hidden surface removal algorithm can be used in this case (OpenGL adopts the z-buffer algorithm)



Surface Culling

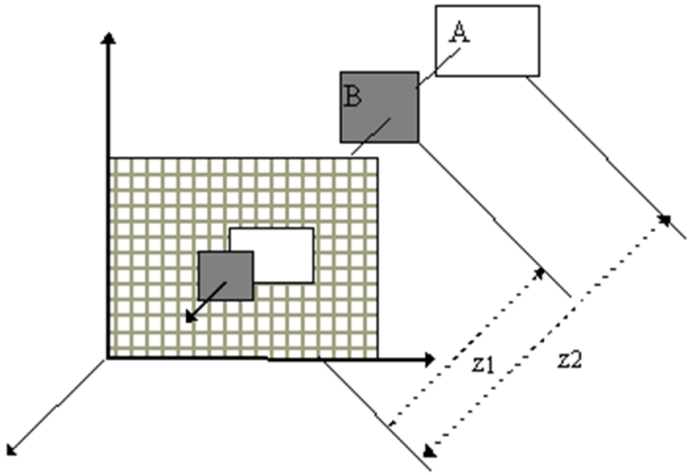
- In order to minimize the amount of work required by the hidden surface removal algorithm, it is possible to remove one specific face of the primitives from the rasterization process (e.g. the back one, the face where the normal vector is applied or the face where the vertices are enumerated in clockwise order)
- If α denotes the angle between the normal vector and the observer direction, the $\cos\alpha > 0$ condition (i.e. $-90° < \alpha < 90°$) defines the primitive's front face
- The elimination of one or both primitive's faces can be enabled by means of the glCullFace. The following example enables the cull face algorithm for the back faces

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

The z-buffer algorithm

- The z-buffer works together with the scan conversion and needs a further buffer called z-buffer to store depth information for each pixel
- The z-buffer is initialized to the distance between the observer and the far clipping plane
- During the scan conversion the distance of the pixel being rendered and the observer can be computed
- If this distance is lower than the distance stored into the z-buffer, then the pixel is visible and the z-buffer is updated with the pixel's depth

The z-buffer algorithm



- When the scan conversion of the polygon B is executed, its color will appear on the screen since its distance z_1 is less than the distance z_2 of the polygon A
- At the contrary, when the scan conversion of the polygon A the pixel corresponding to the intersection point will not appear on the screen

Table of contents

Light

Light

Light-Matter Interaction

Lighting Model

Is a mathematical formulation of the Light Equation

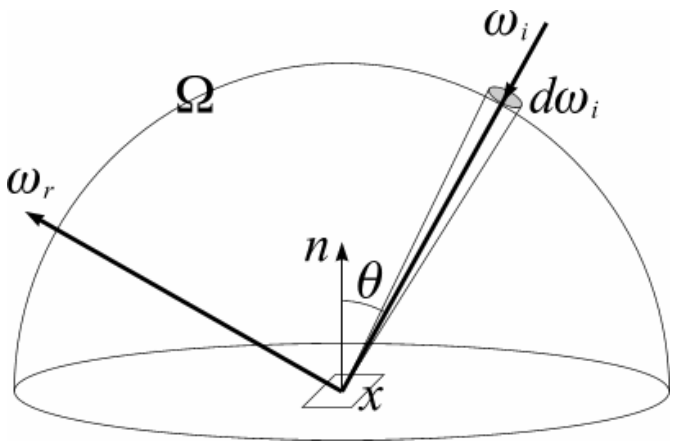
- It is fundamental for photo realistic rendering
- It defines how a point is lit as a function of:
  - position into the three-dimensional space
  - position of the (direct and indirect) sources
  - position of the observer
  - characteristics of the material

The following terms are used in the context of the light equation:

- lighting, which refers to the amount of incident light radiation
- shading, which refers to the evaluation of the resulting color

Radiance Equation

$$L_o(\omega, \overrightarrow{\omega_r}) = L_e(\omega, \overrightarrow{\omega_r}) + L_r(x, \Omega)$$

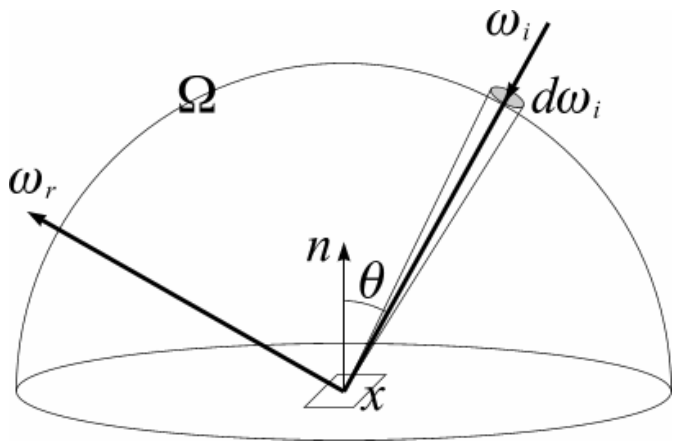


From a position x and direction $\overrightarrow{\omega_r}$, the amount of outgoing light L_o is the sum of the emitted light L_e and of the reflected light L_r

Radiance Equation

L_r is the sum of the lights rays L_i incoming from any direction $\overrightarrow{\omega_i}$ times $\cos\theta$ by the surface's reflection formula f_r

$$L_r = \int_{\Omega} L_i(x, \overrightarrow{\omega_i})(\overrightarrow{\omega_i} \cdot \overrightarrow{n}) f_r(x, \overrightarrow{\omega_r}, \overrightarrow{\omega_r}) d\overrightarrow{\omega_i}$$

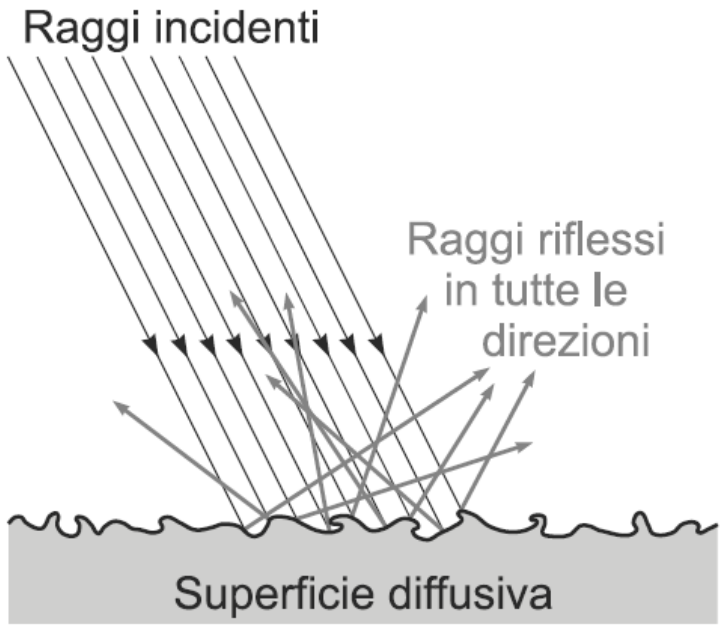


The Phong Model

- The radiance equation is impossible to be evaluated in real time with current hardware
- OpenGL, which is a real time API for 3D Computer Graphics, adopts the simplified Phong Model where:
  - Only directional and point light sources are considered
  - No inter-reflections are taken into account
  - The light equation is evaluated locally, thus possible occluders are ignored
  - f_r is approximated with three constants (that are used to characterize the material of the reflecting surface)
- As a consequence, the only simulated phenomenon is the (specular and diffuse) reflection

The Phong Model: diffuse reflection

- The incoming rays are uniformly reflected in any direction
- The amount of light does not depend on the position of the observer and is proportional to the cosine of the angle between the incoming light and the surface normal (Lambert's law)



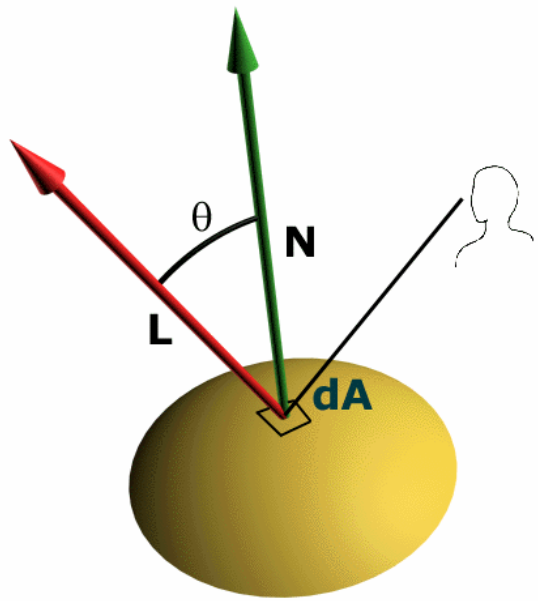
The Phong Model: diffuse reflection

Phong's diffuse equation

$$I_{diff} = I_p k_d \cos\theta = I_p k_d (\overrightarrow{N} \cdot \overrightarrow{L})$$

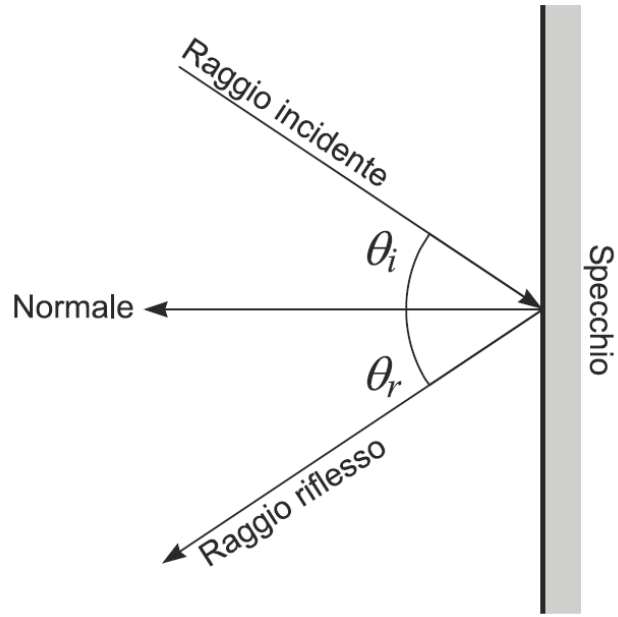
Depends on:

- Surface orientation, $\overrightarrow{N}$
- Light direction, $\overrightarrow{L}$
- The surface's reflection function, which is approximated by the constant k_d



The Phong Model: specular reflection

- The angle between the reflected light and the surface normal is equal to the angle between the incoming light and the normal ($\theta_i = \theta_r$)
- As a consequence, the amount of light depends on the position of the observer

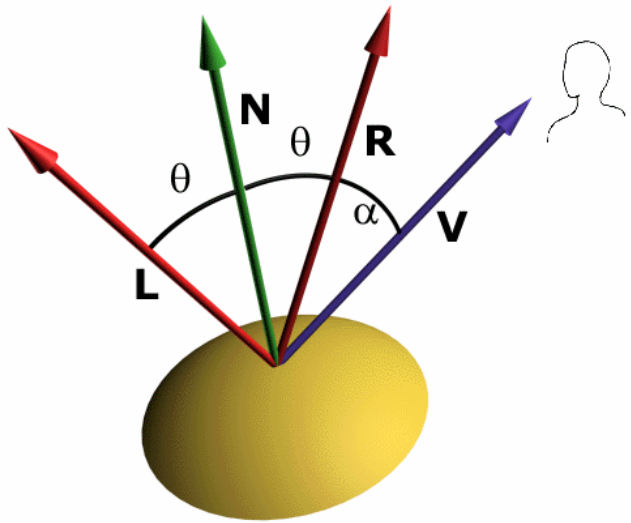


Phong's specular equation

Specular Equation

$$I_{spec} = I_p k_s \cos^n \alpha = I_p k_s (\vec{R} \cdot \vec{V})^n$$

where k_s is the constant used to approximate the surface's specular reflection function



The Phong Model: specular reflection

- The specular reflection is perceived as a spot highlight that depends on the position of the observer
- The highest highlight is obtained when $\alpha = 0$ and decreases rapidly according to the law

$$(\cos\alpha)^n$$

, where $n \in [1, 128]$ is called *specular reflection exponent*

Phong ambient equation

It is used to model the inter-reflections

Phong's ambient equation

$$I_{amb} = I_a k_a$$

where k_a is a material dependent constant

Complete formulation of the Phong Model

Complete formulation of the Phong Mode

$$I = I_a k_a + \sum_p I_p \left(k_d (\overrightarrow{N} \cdot \overrightarrow{L}) + k_s (\overrightarrow{R} \cdot \overrightarrow{V})^n \right)$$

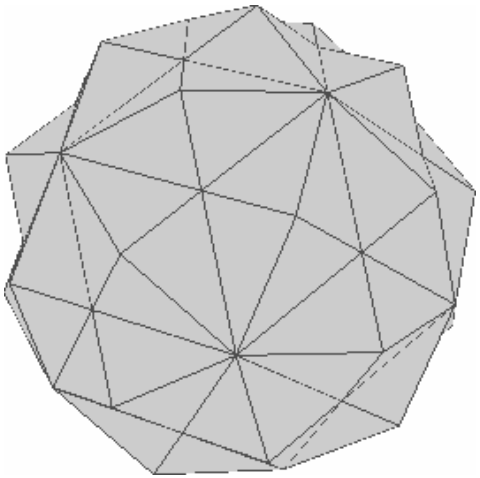
The model can also account for the attenuation due to the distance by means of an attenuation factor

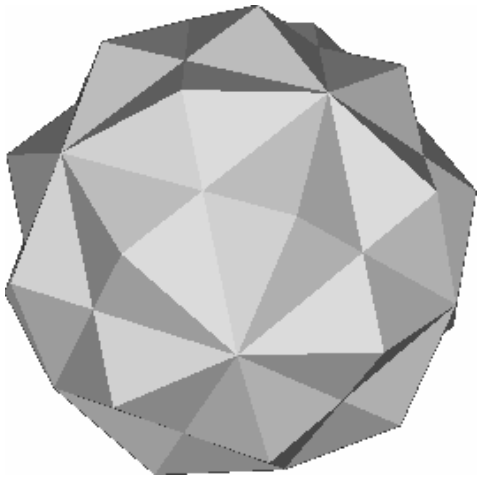
Shading

- The color of a fragment is evaluated by applying a shading model
- The best shading model would consist in calculating the light equation for each pixel of the final image
- However, approximate solutions have to be adopted to obtain real time applications, especially on old hw

Flat Shading

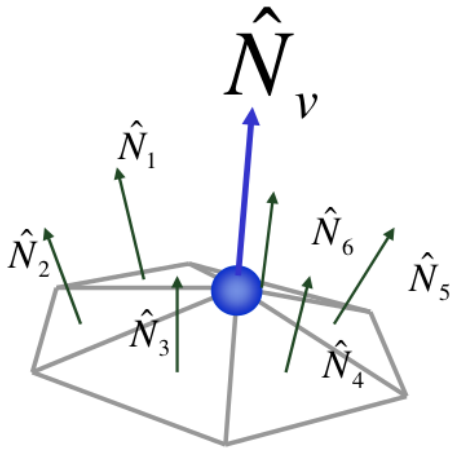
- The flat shading, is the simplest shading model consisting into evaluating the light equation once for the whole geometric primitive
- The same color is thus considered for each fragment of the primitive



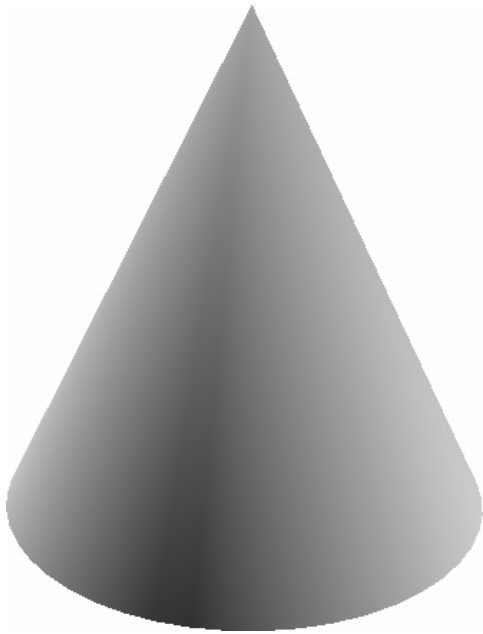


Gouraud shading

- Exploits the linearity of the RGB color space: the color of an inner fragment is obtained by linearly interpolating the colors of the vertices
- The light equation is thus evaluated for the vertices only
- Both the surface's normal (if known) or the average of the normal vectors evaluated for the vertex (since it is shared by more than one geometric primitive) can be considered

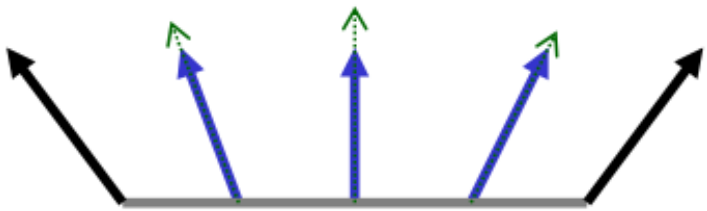


Gouraud shading



Phong shading

- It is the more sophisticated shading model and is the most used in the case of surfaces with a high specular reflection
- In spite of interpolating colors, the Phong shading interpolates the normals
- The lighting equation is evaluated for all the inner fragments of the geometric primitive by considering the interpolated normal vectors



Comparison of different shading models