Computer Graphics

An Introduction

What's this course all about?

We will cover…

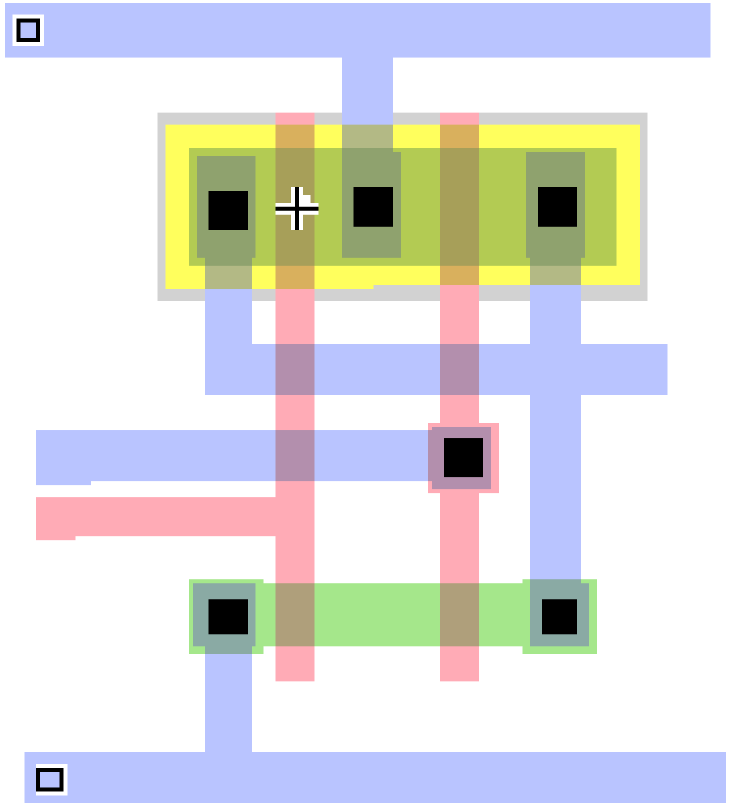
- Graphics programming and algorithms
- Scanconversion algorithms
- Color and Shading
- Applied geometry (Curves and surfaces)
- modelling and rendering

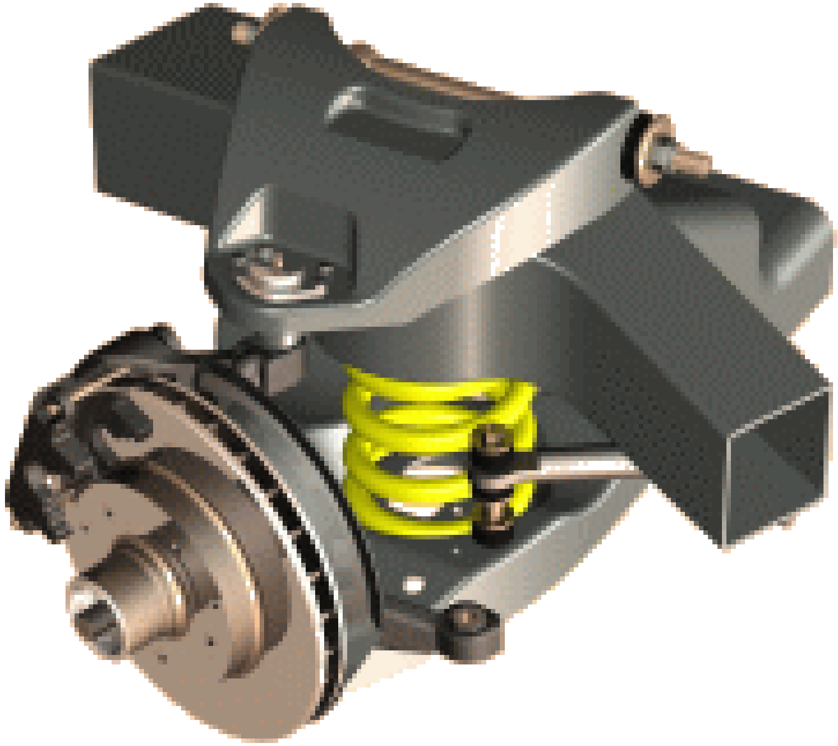
Computer Graphics is about animation (films)

Games are very important in Computer Graphics

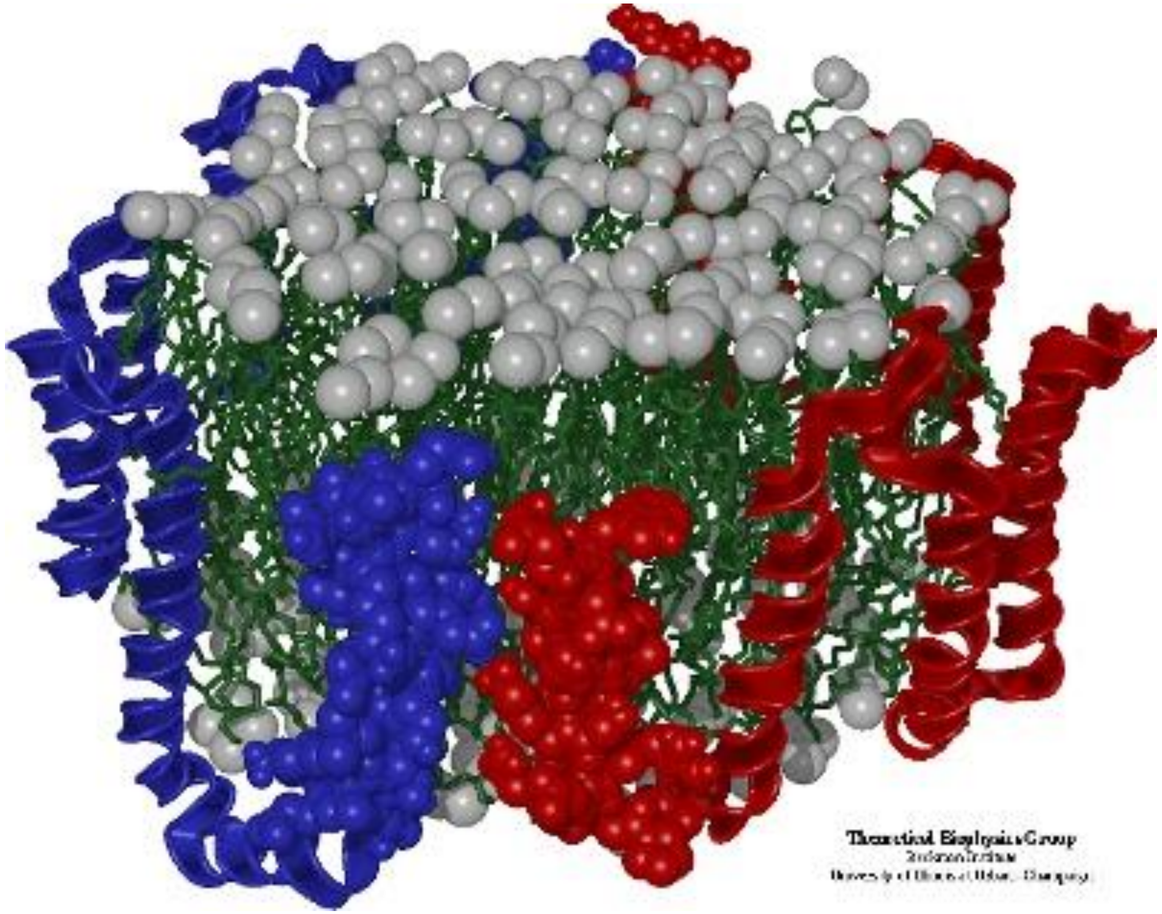
Medical Imaging is another driving force

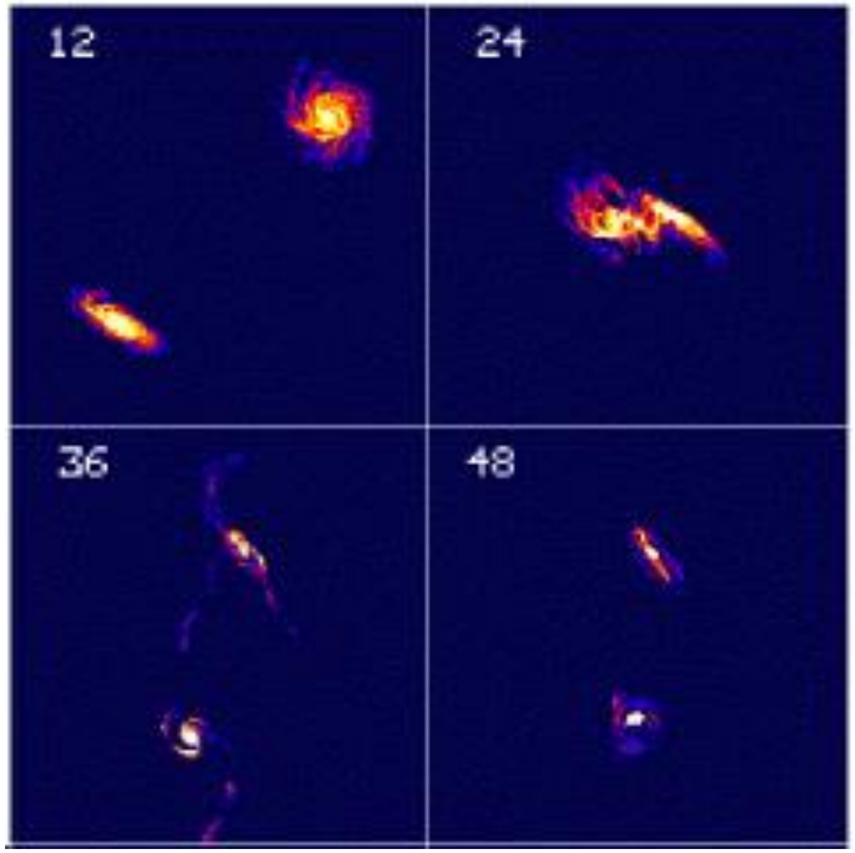
Computer Aided Design too





Scientific Visualisation





Overview of the Course

Graphics Pipeline (Today)

Modelling

  Surface / Curve modelling

(Local lighting effects) Illumination, lighting, shading, mirroring, shadowing

Rasterization (creating the image using the 3D scene)

Ray tracing

Global illumination

Curves and Surfaces

Graphics/Rendering Pipeline

1. Graphics processes generally execute sequentially

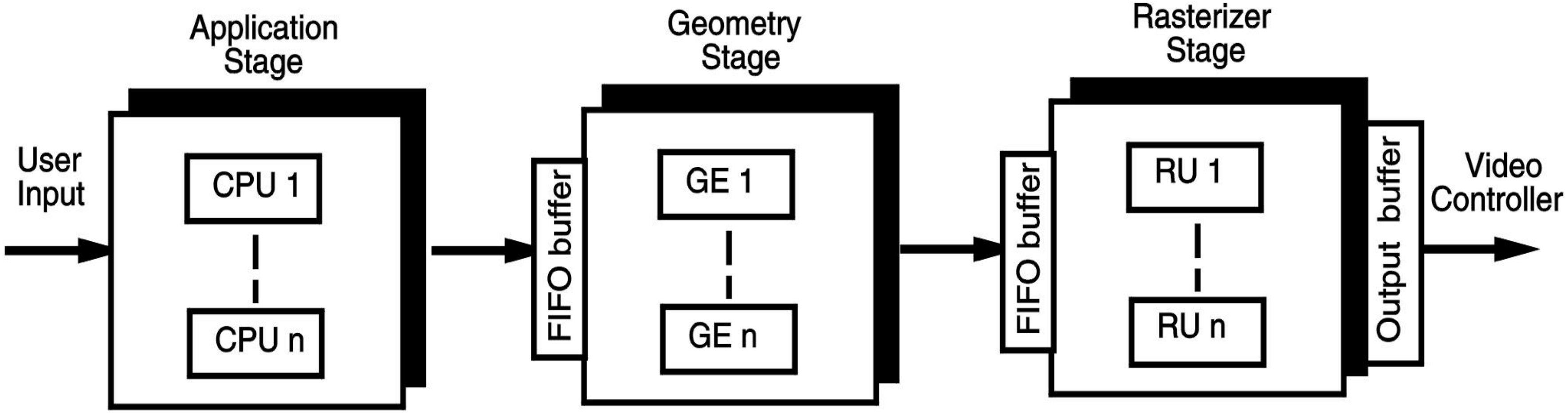
2. Pipelining the process means dividing it into stages

3. Especially when rendering in real-time, different hardware resources are assigned for each stage

Graphics / Rendering Pipeline

There are three stages

1. Application Stage
2. Geometry Stage
3. Rasterization Stage



Application stage

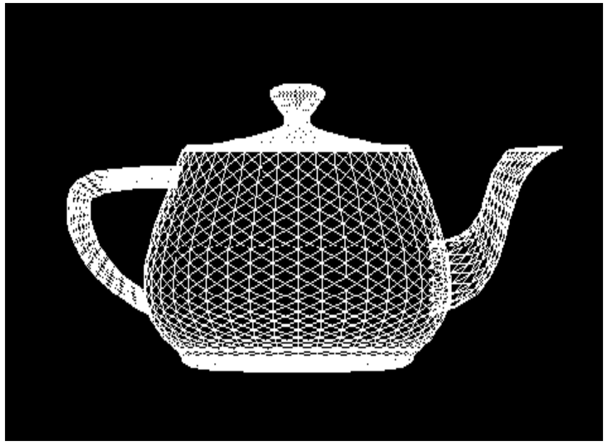
Entirely done in software by the CPU

Read Data

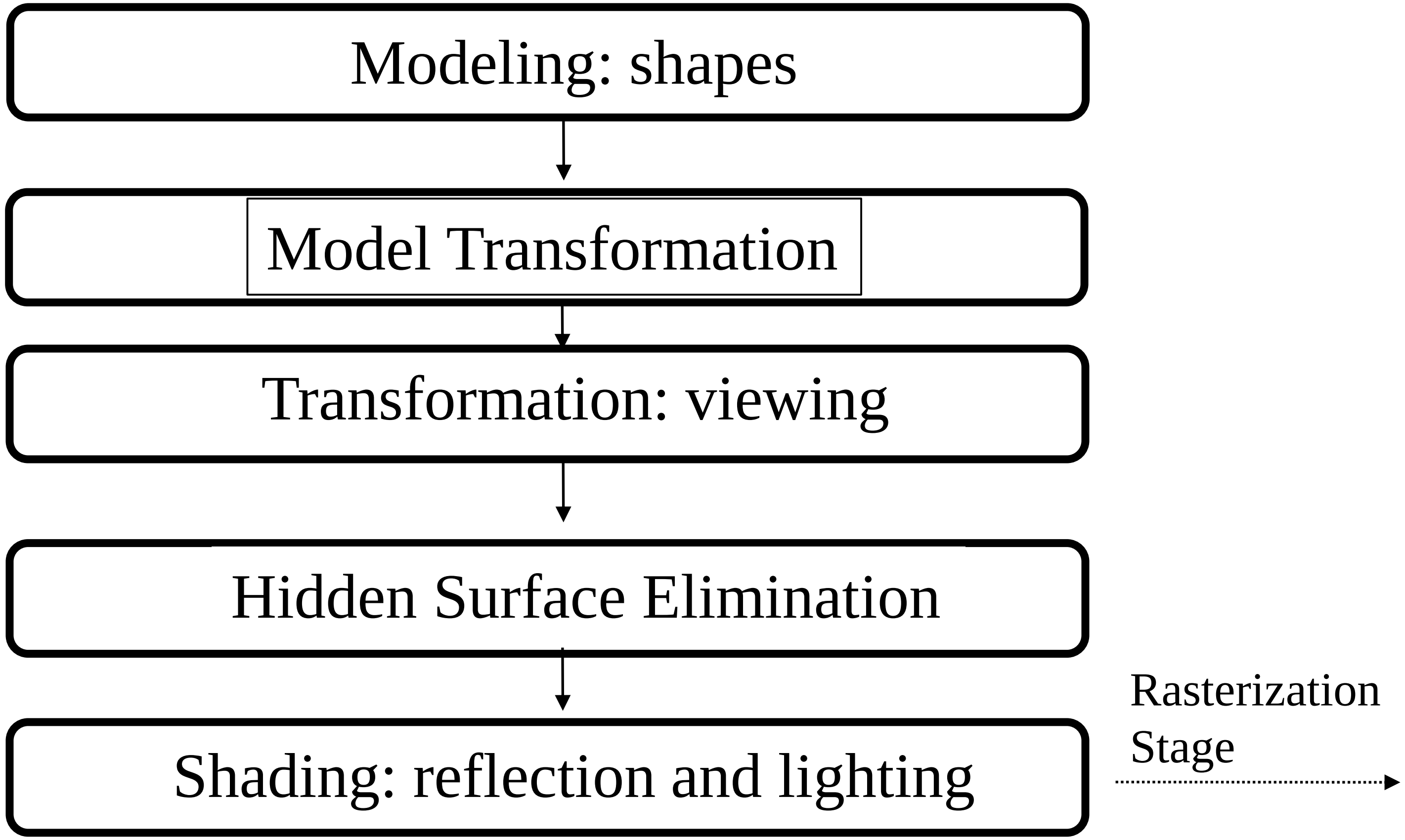
- the world geometry database,
- User’s input by mice, trackballs, trackers, or sensing gloves

In response to the user’s input, the application stage change the view or scene

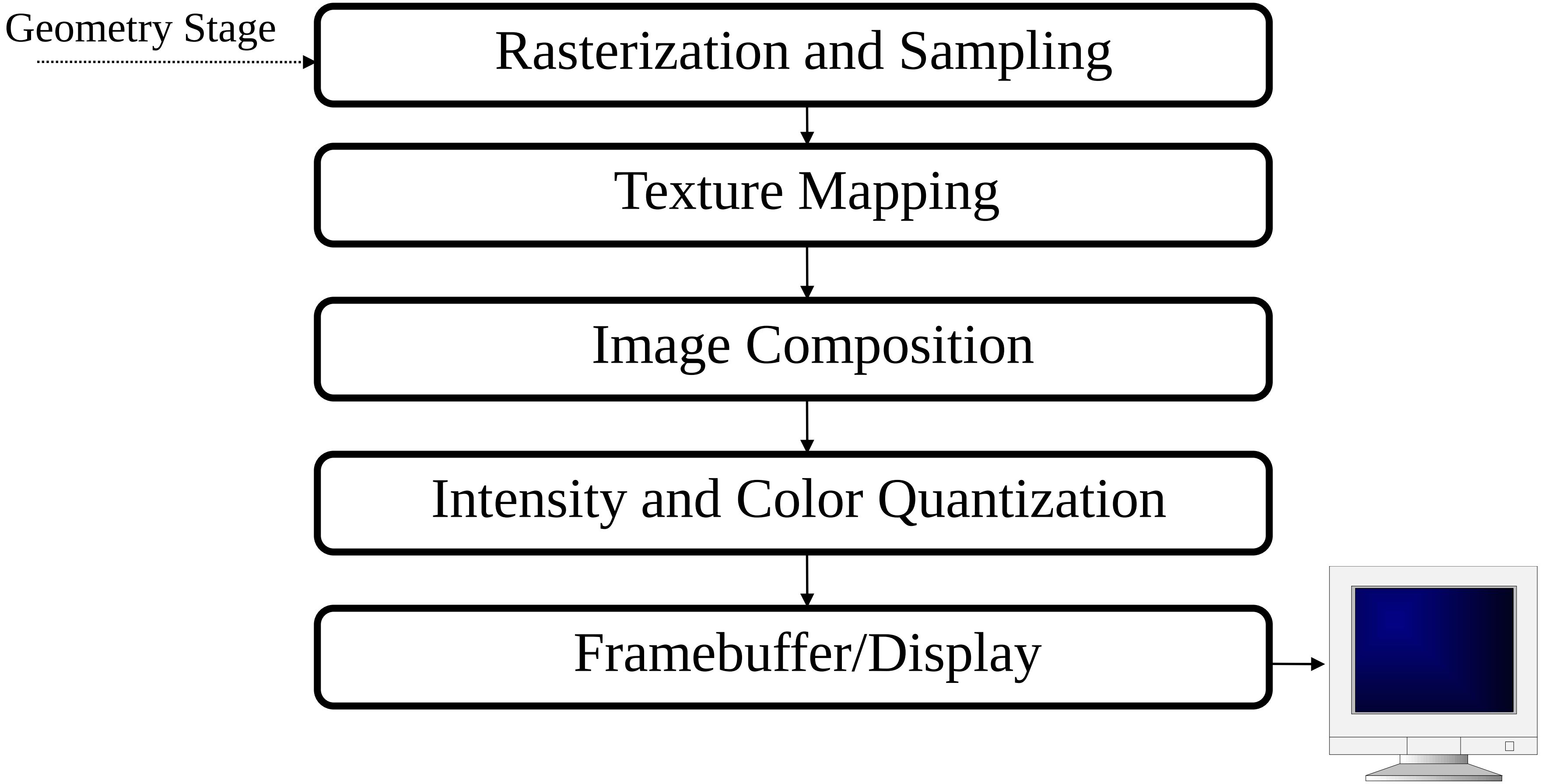
```
v 3.382035 2.446498 -0.064692
v 3.382035 2.446498 0.064692
v 3.392006 2.474995 -0.050004
v 3.392006 2.474995 0.050004
v 3.400000 2.446800 0.000000
v 3.406947 2.462176 -0.061668
v 3.406947 2.462176 0.061668
v 3.408000 2.475600 0.000000
v 3.408000 2.475600 0.000000
v 3.411237 2.476753 -0.054000
v 3.411237 2.476753 0.054000
v 3.416450 2.472371 -0.057996
v 3.416450 2.472371 0.057996
v 3.424875 2.462606 0.000000
v 3.428125 2.477344 0.000000
v 3.428125 2.477344 0.000000
v 3.434000 2.472900 0.000000

2909 2921 2939
2939 2931 2909
2869 2877 2921
2921 2909 2869
2819 2827 2877
2877 2869 2819
2737 2747 2827
2827 2819 2737
2669 2673 2747
2747 2737 2669
2567 2575 2673
2673 2669 2567
```



Geometry Stage

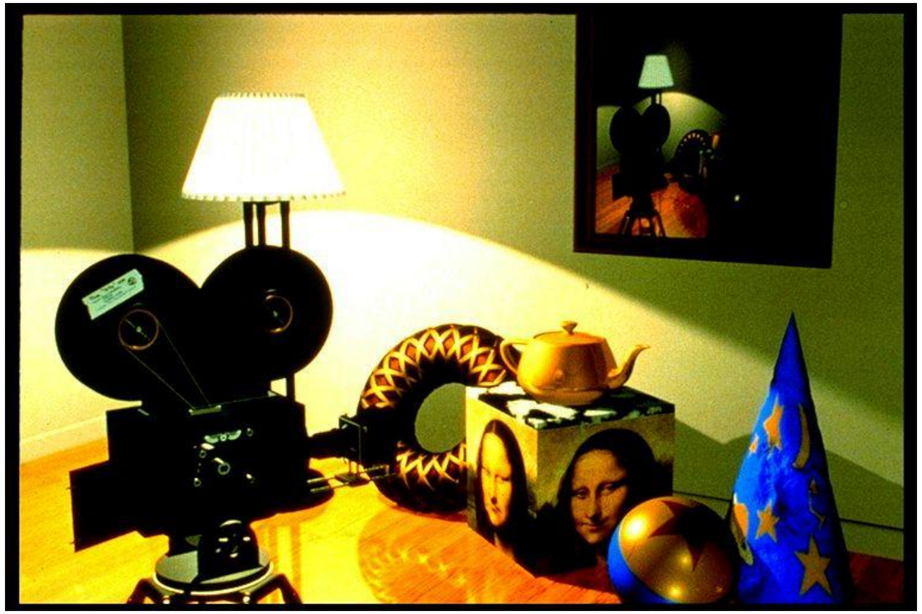


Rasterization Stage

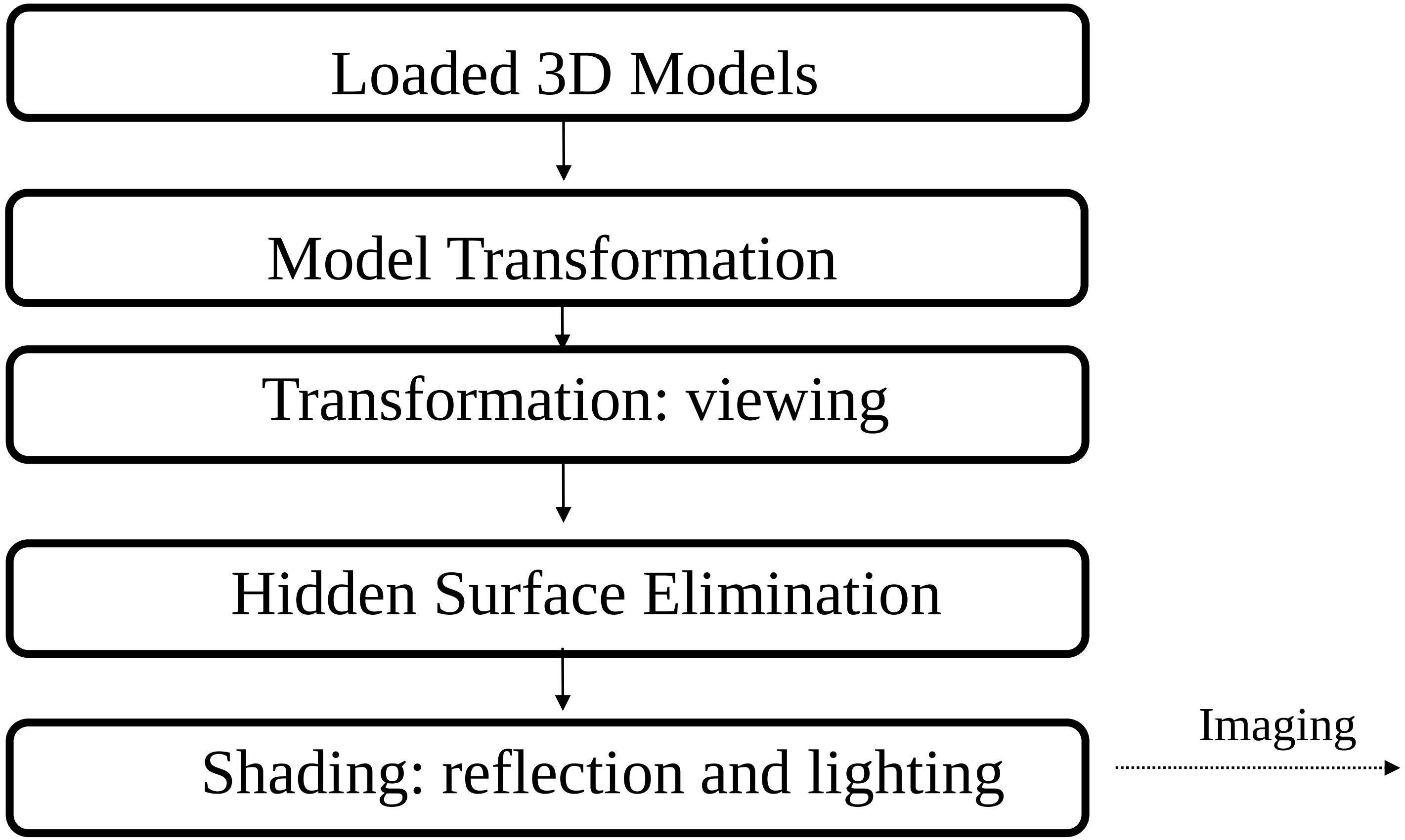


An example thro' the pipeline…

The scene we are trying to represent:



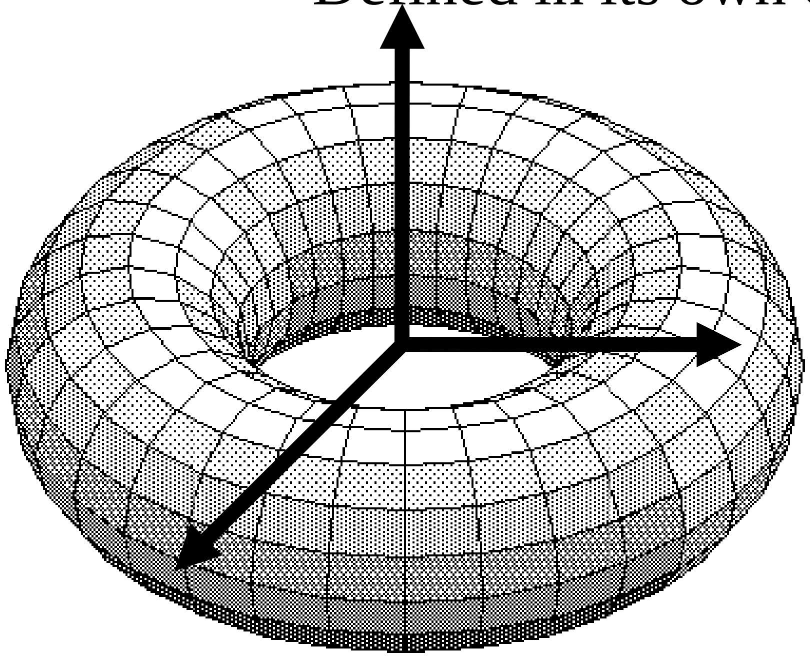
Geometry Pipeline

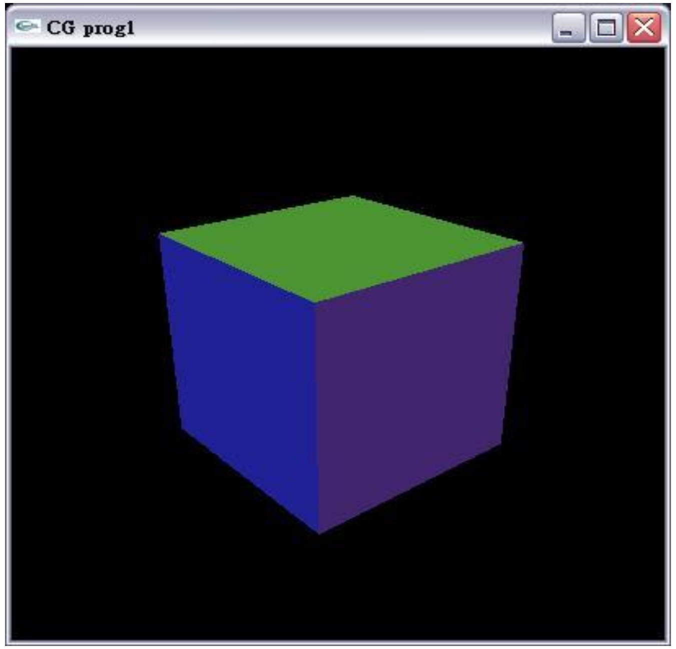


Preparing Shape Models

Designed by polygons, parametric curves/surfaces, implicit surfaces and etc.

Defined in its own coordinate system



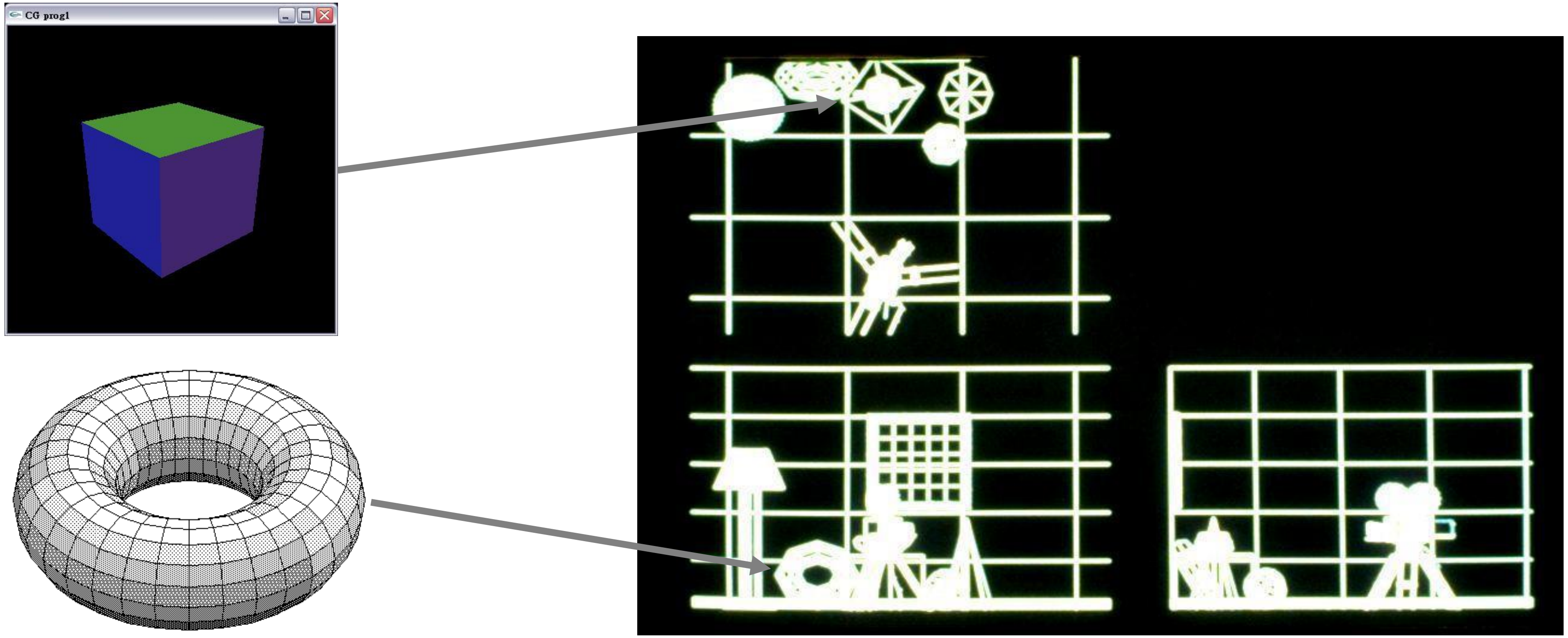


Model Transformation

Objects put into the scene by applying translation, scaling and rotation

Linear transformation called homogeneous transformation is used

The location of all the vertices are updated by this transformation

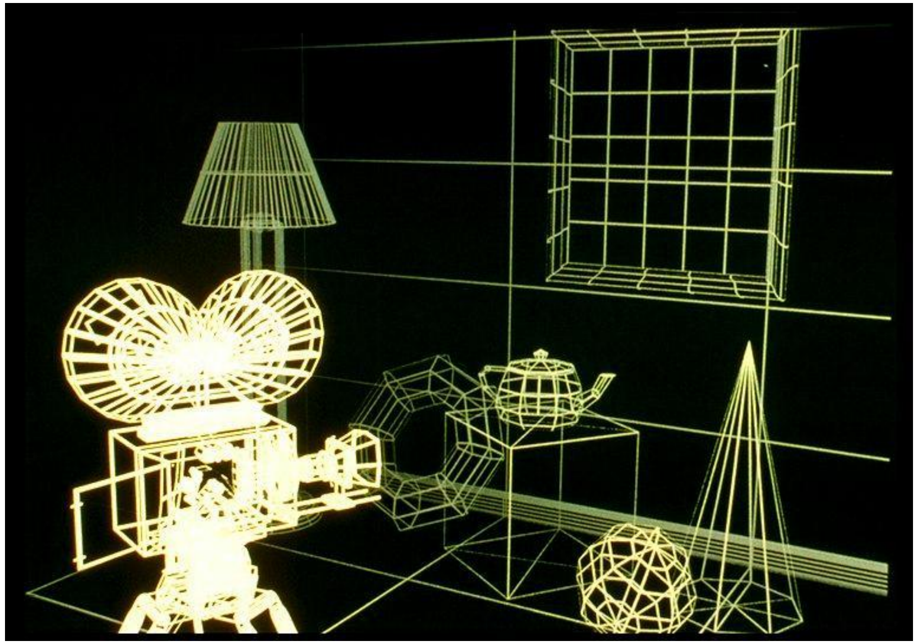


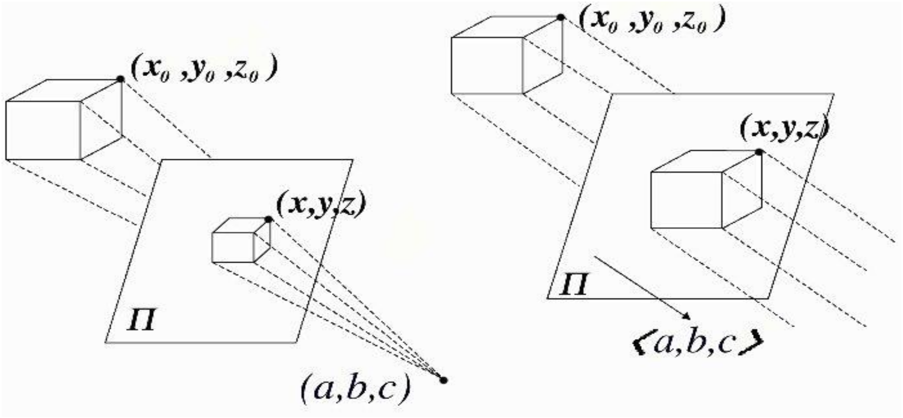
Perspective Projection

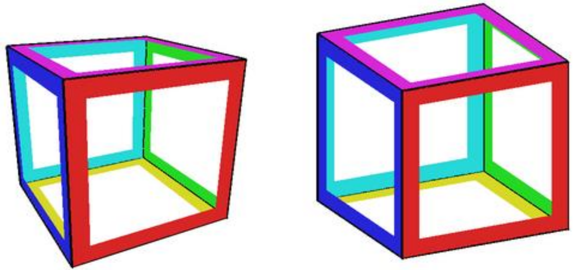
We want to create a picture of the scene viewed from the camera

We apply a perspective transformation to convert the 3D coordinates to 2D coordinates of the screen

Objects far away appear smaller, closer objects appear bigger

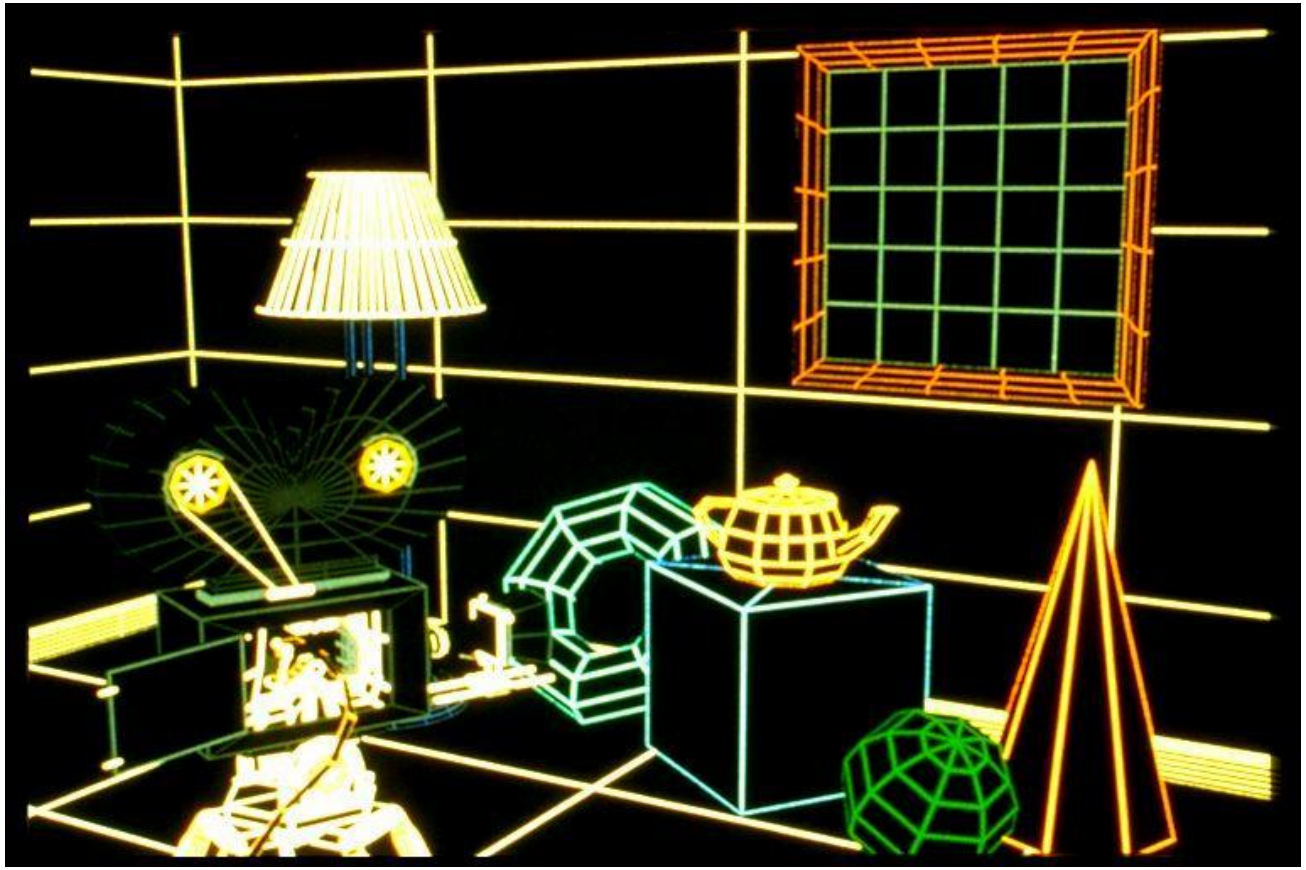






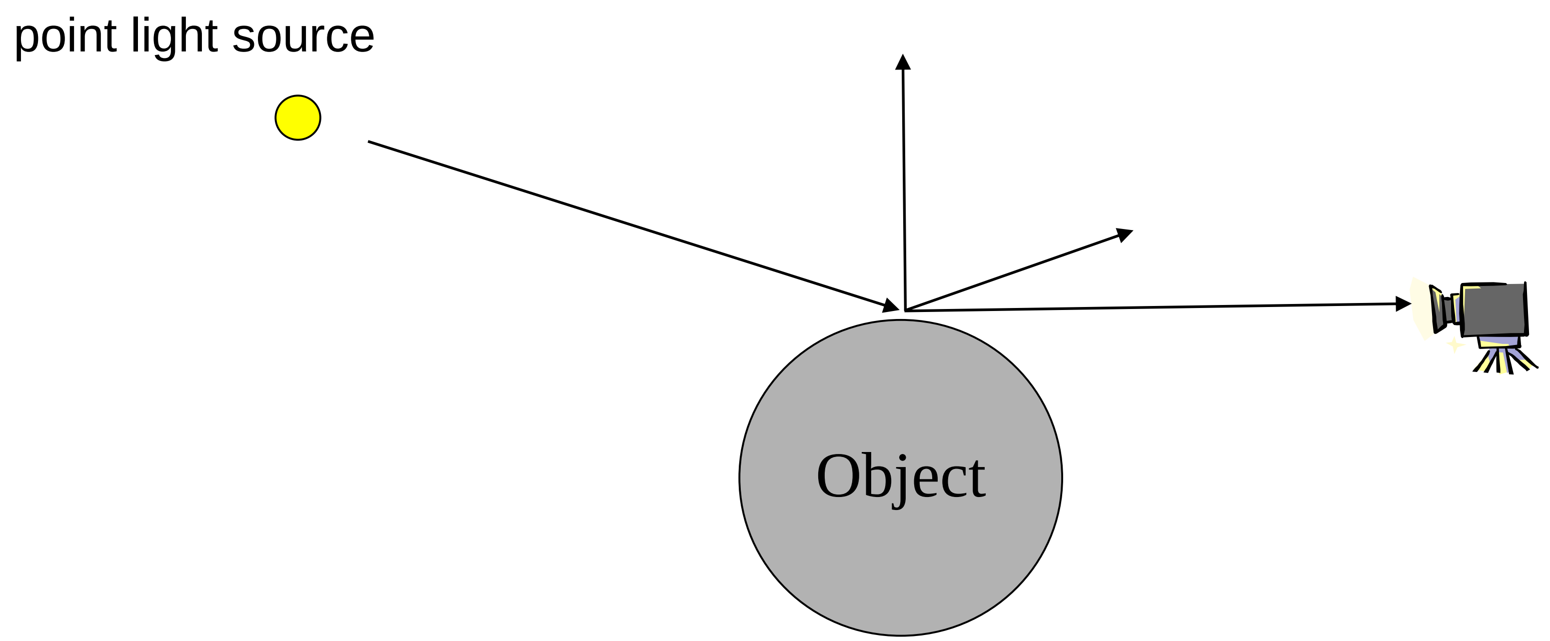
Hidden Surface Removal

Objects occluded by other objects must not be drawn



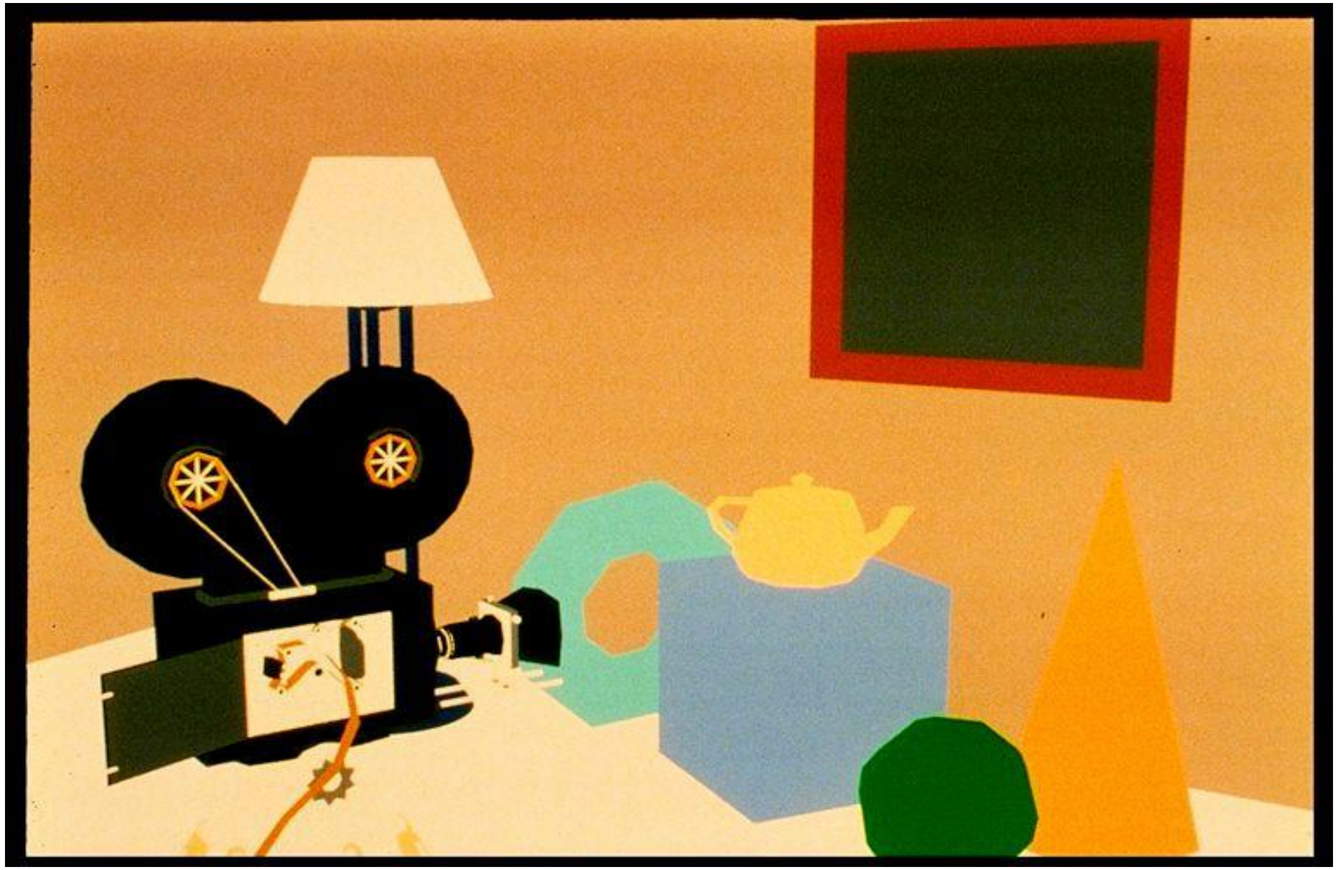
Shading

Now we need to decide the colour of each pixels taking into account the object’s colour, lighting condition and the camera position



Shading : Constant Shading - Ambient

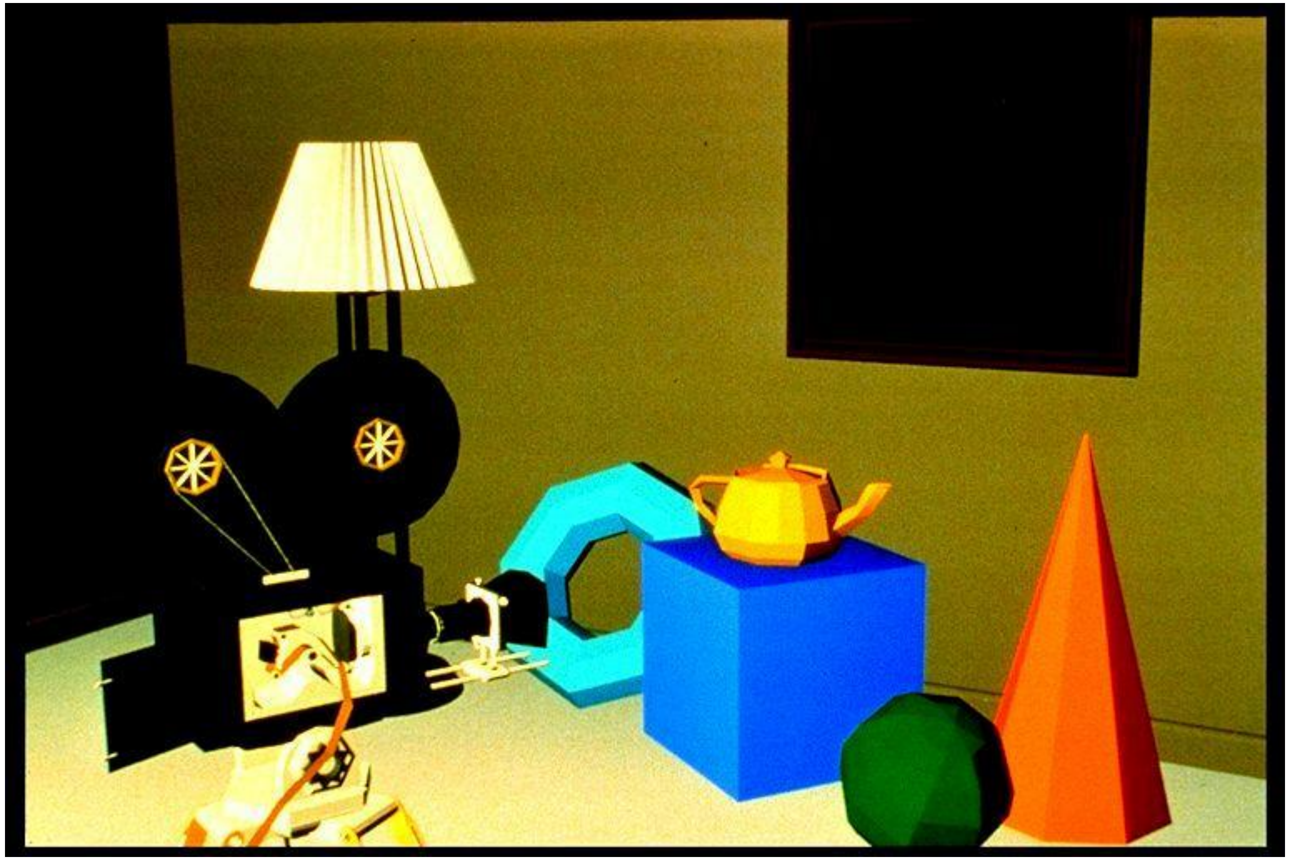
Objects colours by its own colour



Shading – Flat Shading

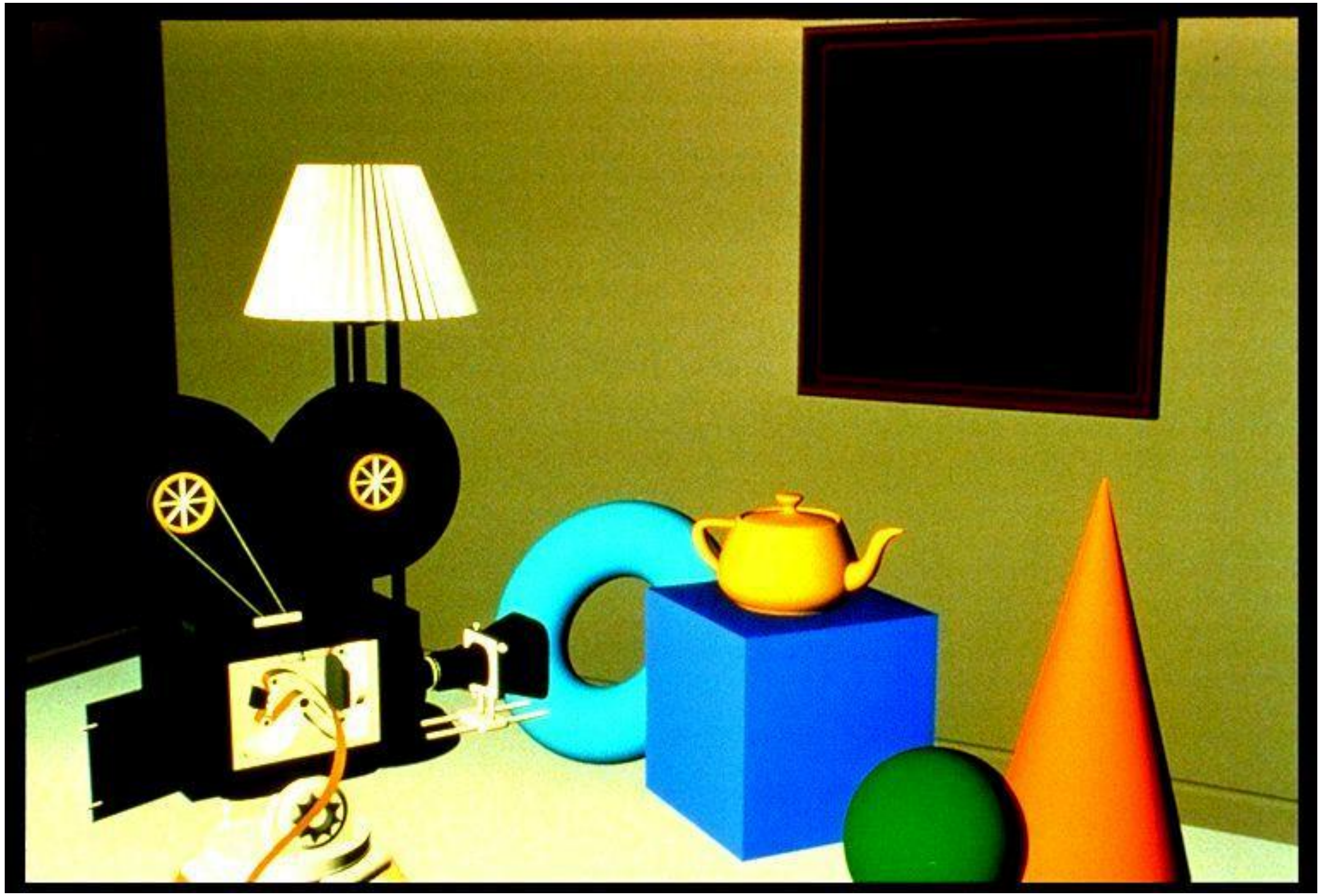
Objects colored based on its own colour and the lighting condition

One colour for one face

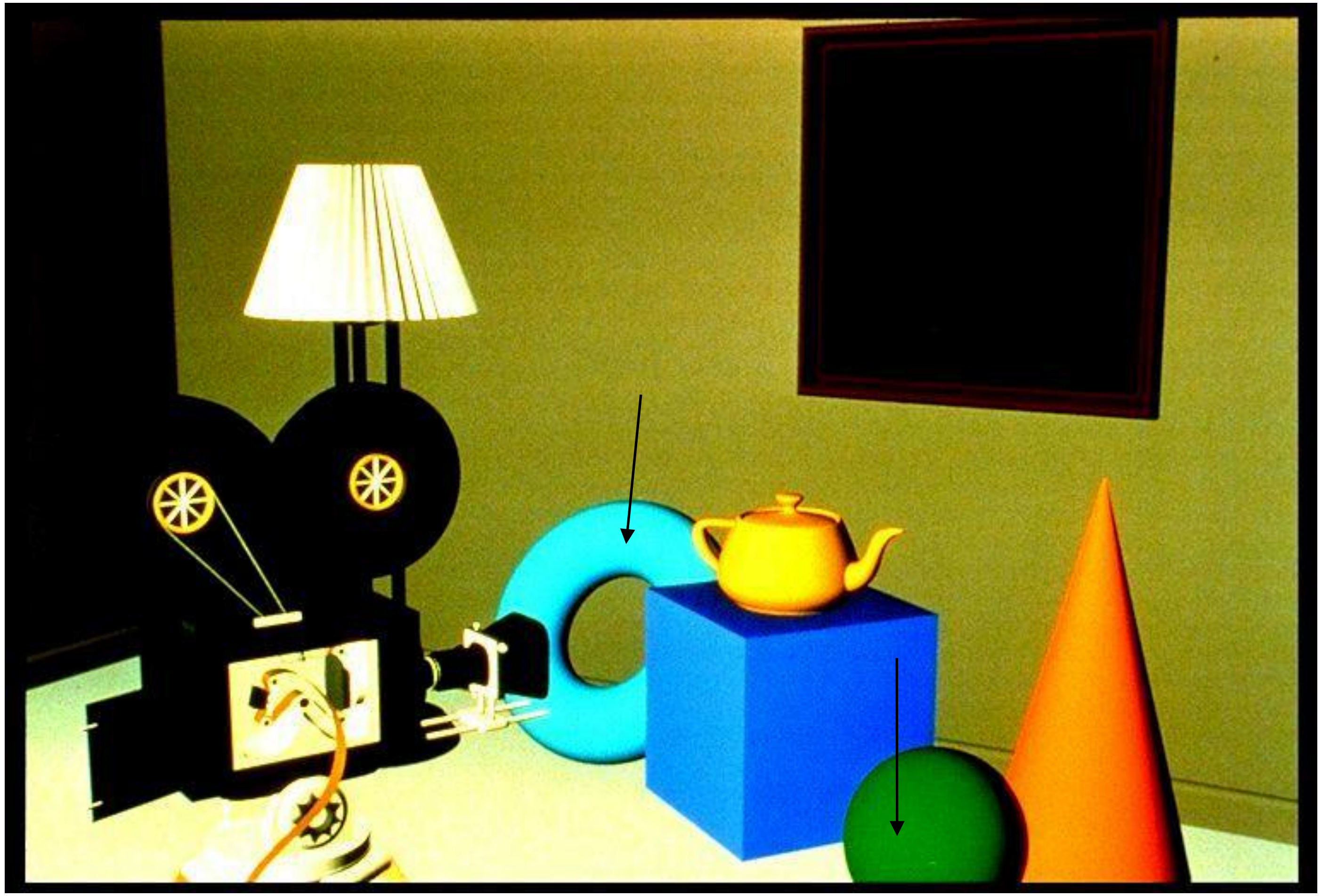


Gouraud shading, no specular highlights

Lighting calculation per vertex

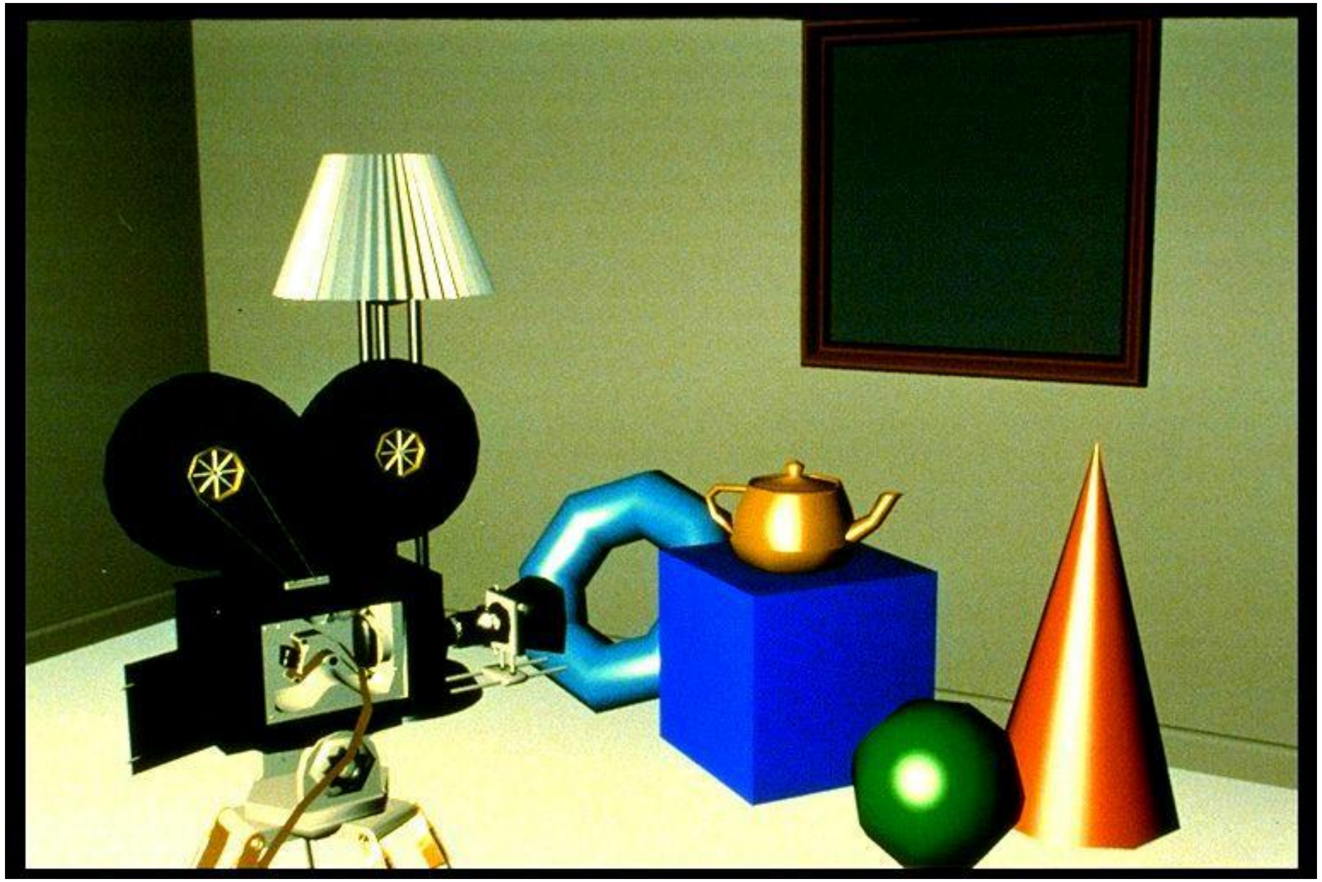


Shapes by Polynomial Surfaces

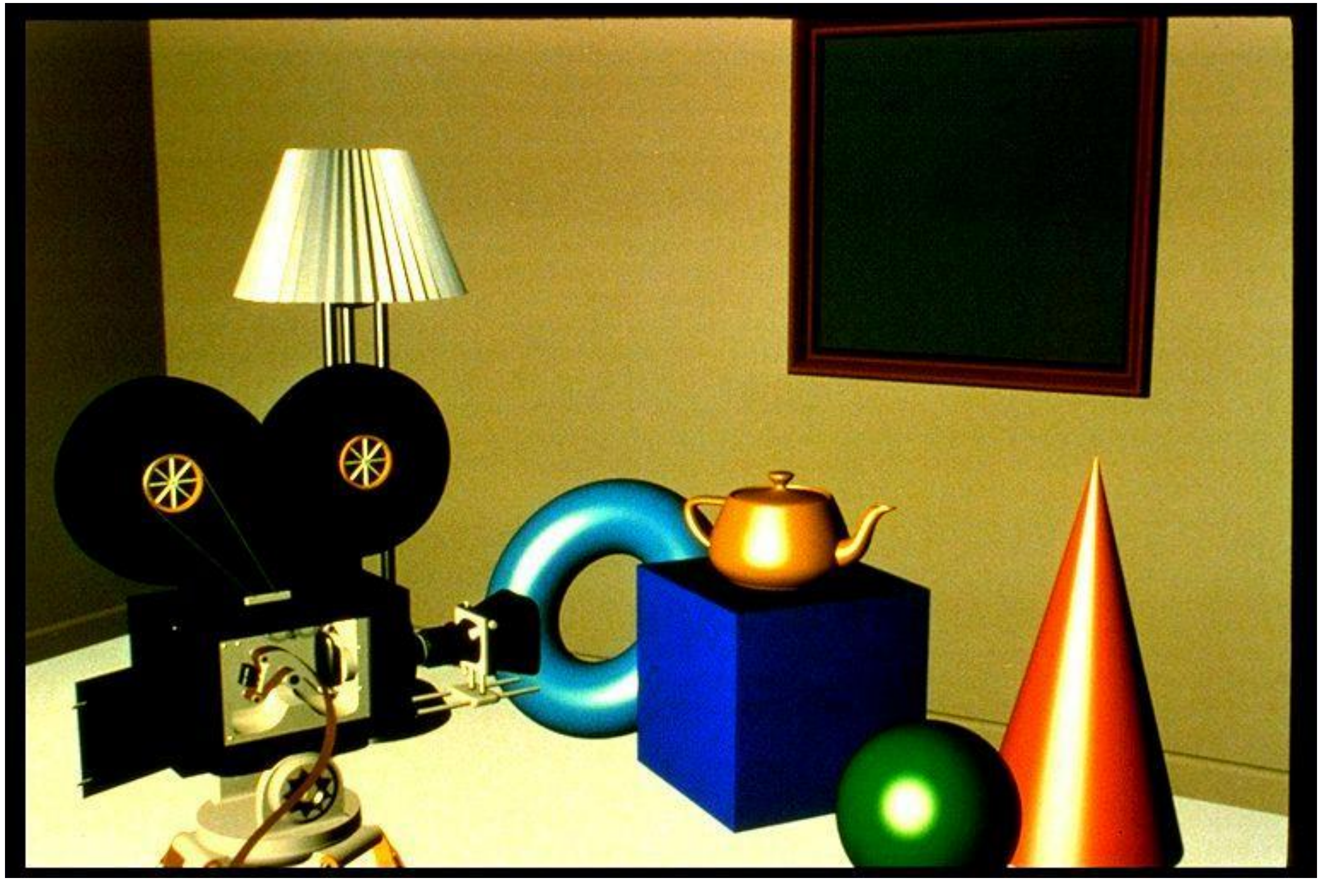


Specular highlights added

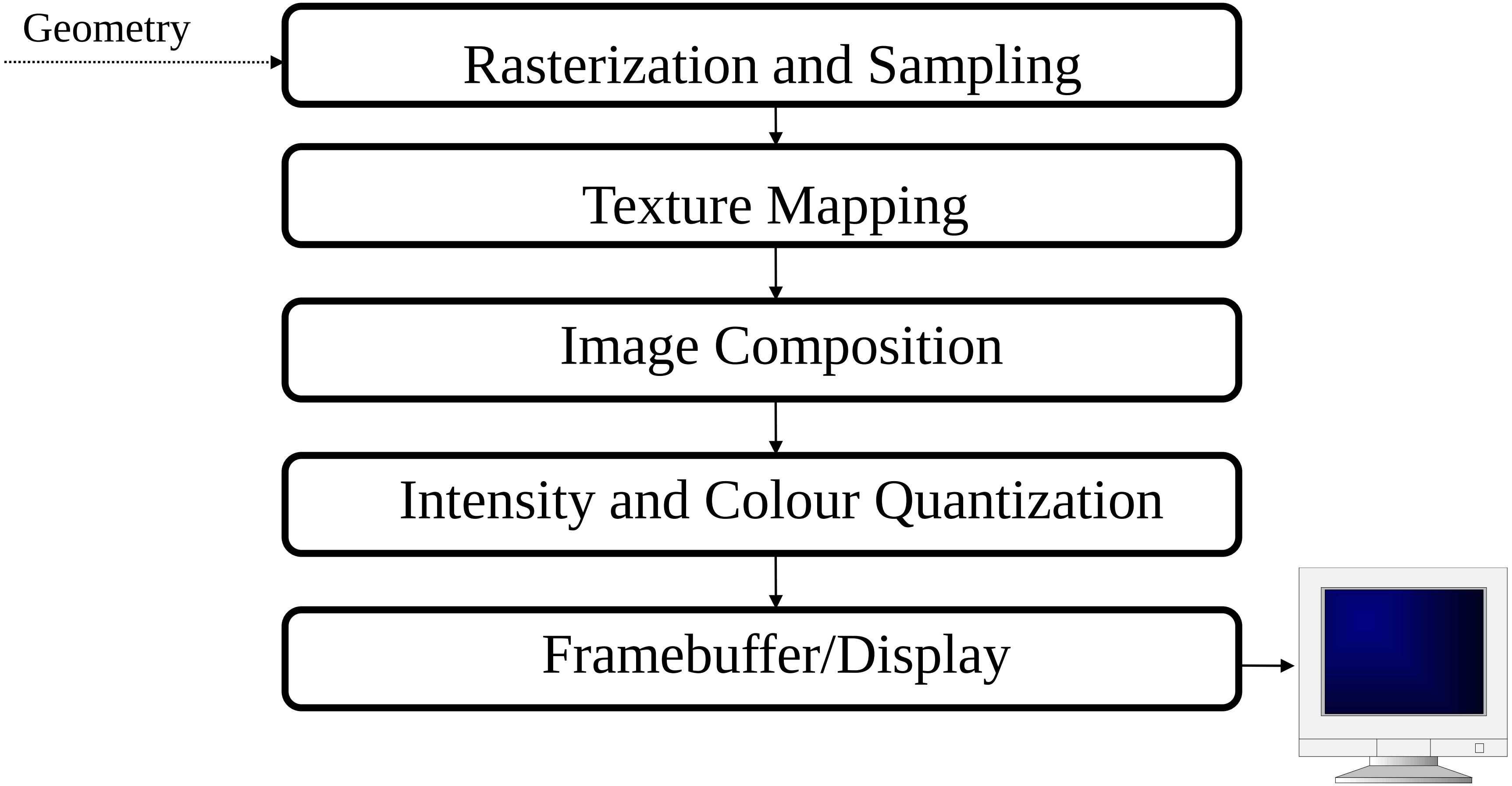
Light perfectly reflected in a mirror-like way



Phong shading



Next, the Imaging Pipeline

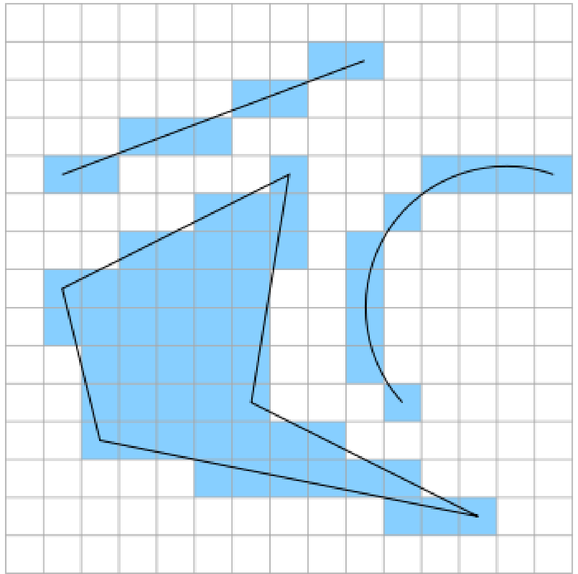


Rasterization

Converts the vertex information output by the geometry pipeline into pixel information needed by the video display

Aliasing: distortion artifacts produced when representing a high-resolution signal at a lower resolution.

Anti-aliasing : technique to remove aliasing

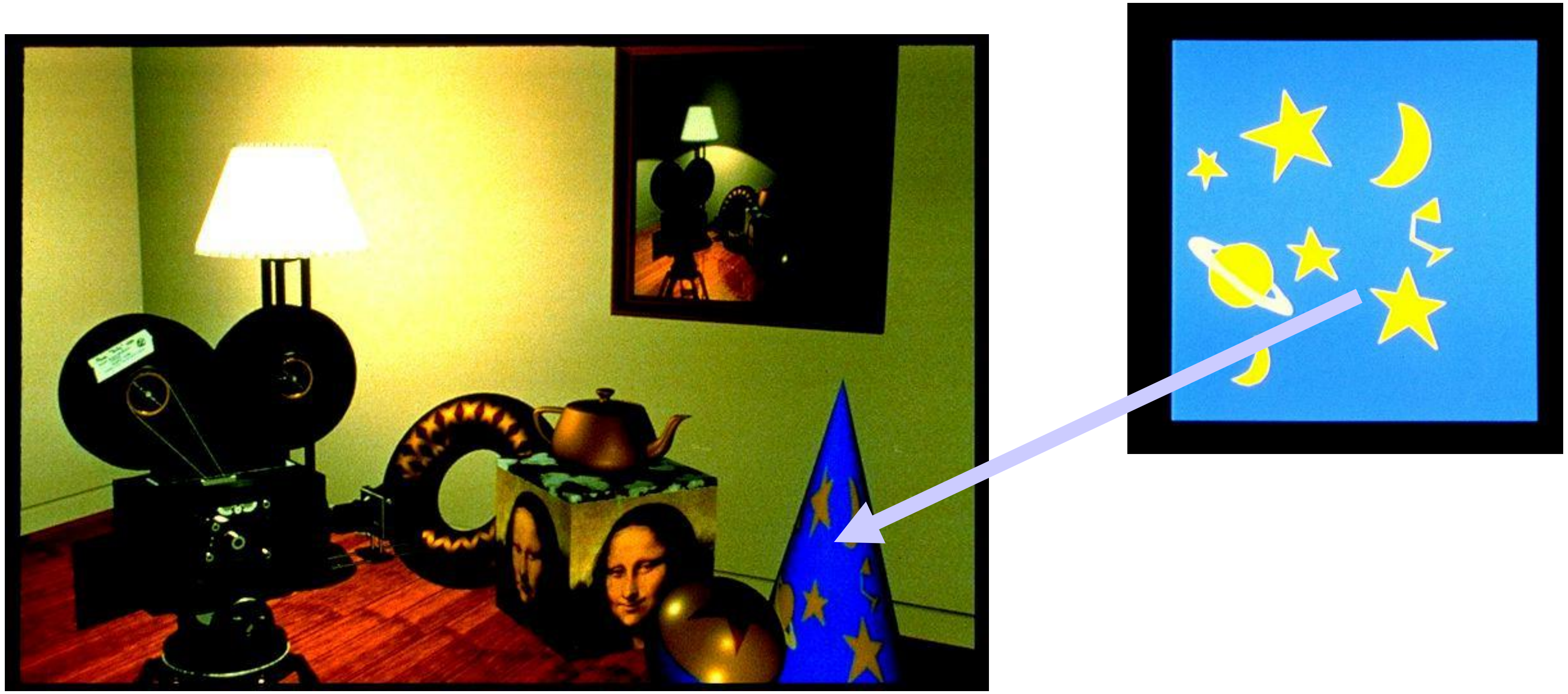


Anti-aliasing

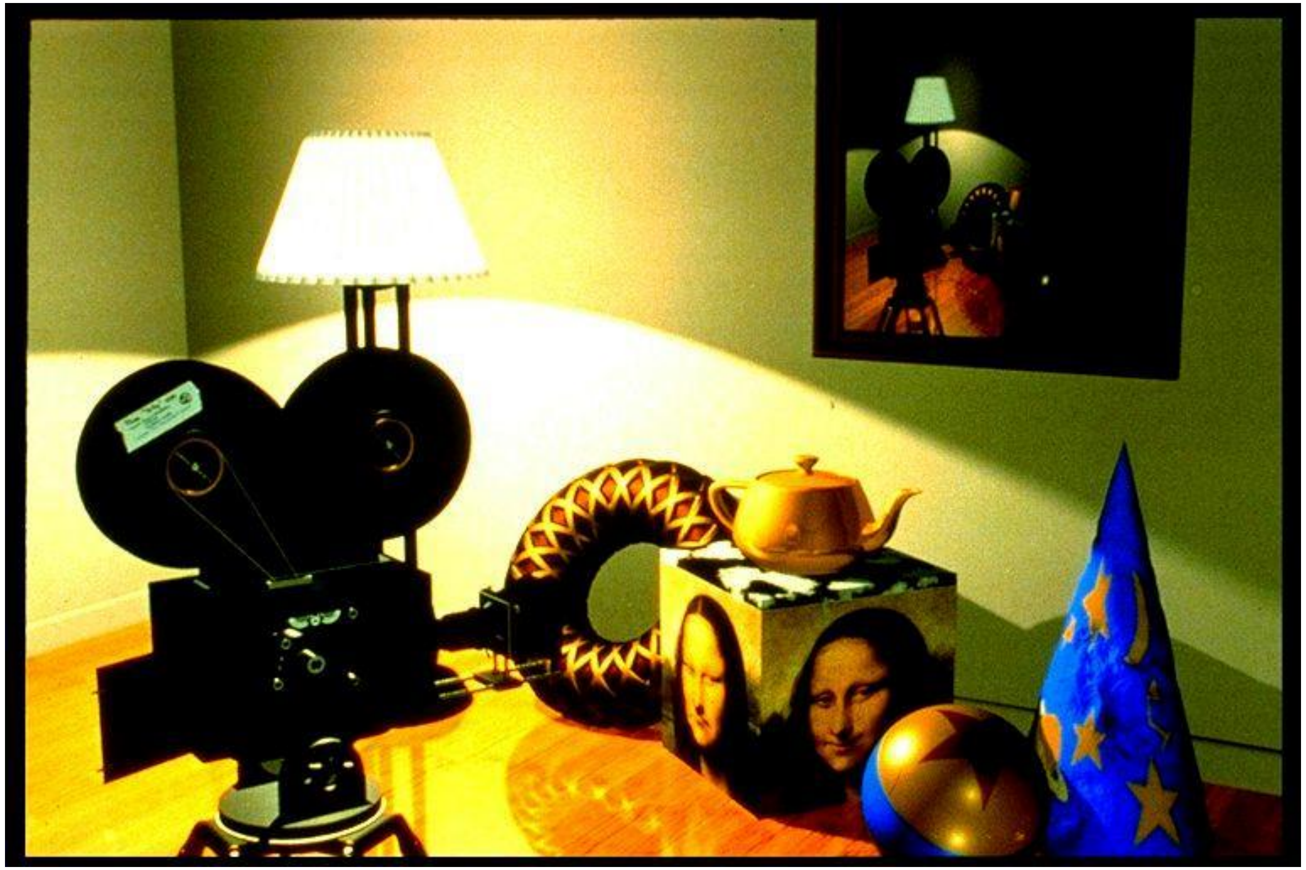
Aliased polygons
(jagged edges)

Anti-aliased polygons

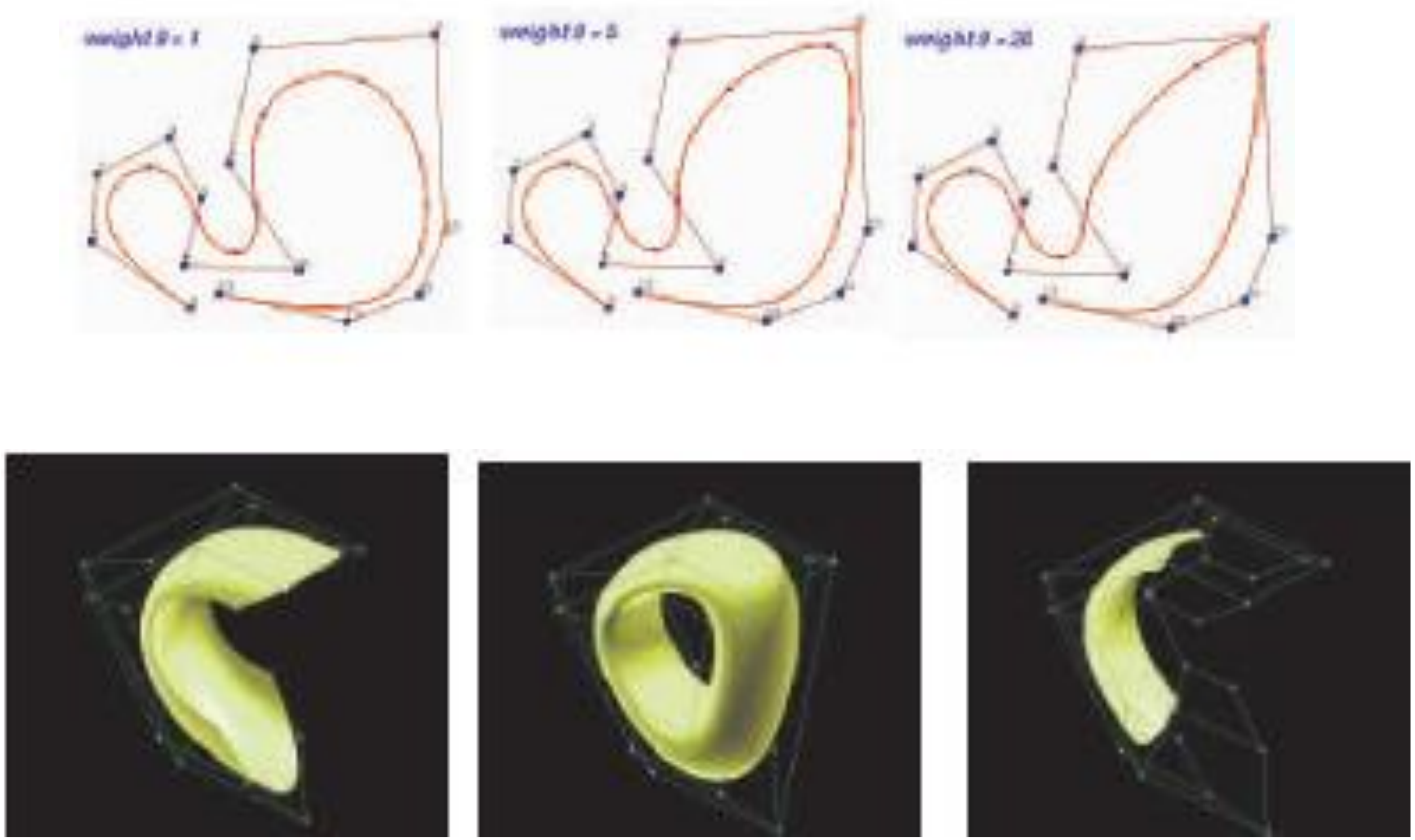
Texture mapping



Other covered topics:
Reflections, shadows & Bump mapping



Polynomial Curves, Surfaces

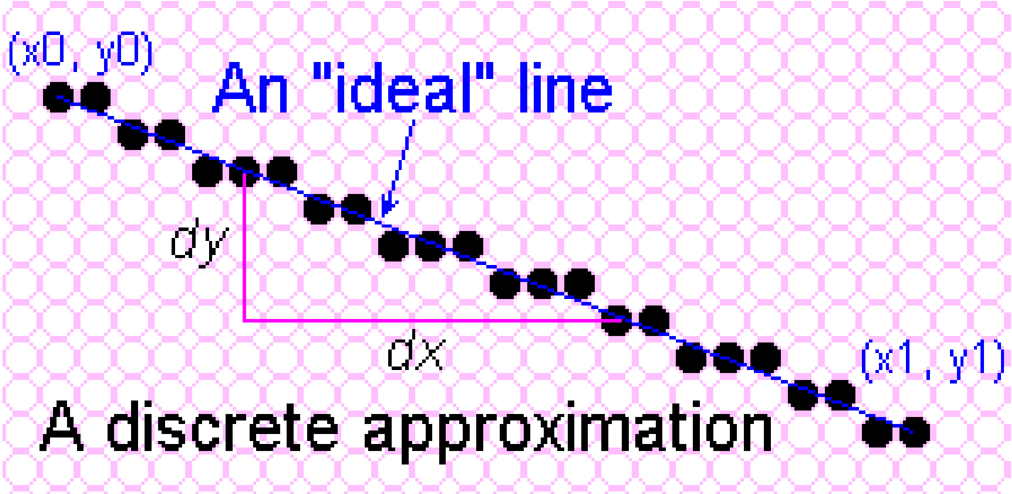


Summary

- The course is about algorithms, not applications
  - Lots of mathematics
- Graphics execution is a pipelined approach
- Basic definitions presented
- Some support resources indicated

Towards the Ideal Line

- We can only do a discrete approximation



- Illuminate pixels as close to the true path as possible, consider bi-level display only
  - Pixels are either lit or not lit

What is an *ideal* line

- Must appear straight and continuous (?)
  - Only possible axis-aligned and 45^{o} lines
- Must interpolate both defining end points
- Must have uniform density and intensity
  - Consistent within a line and over all lines
  - What about antialiasing (?)
- Must be efficient, drawn quickly
  - Lots of them are required!!!

Simple Line

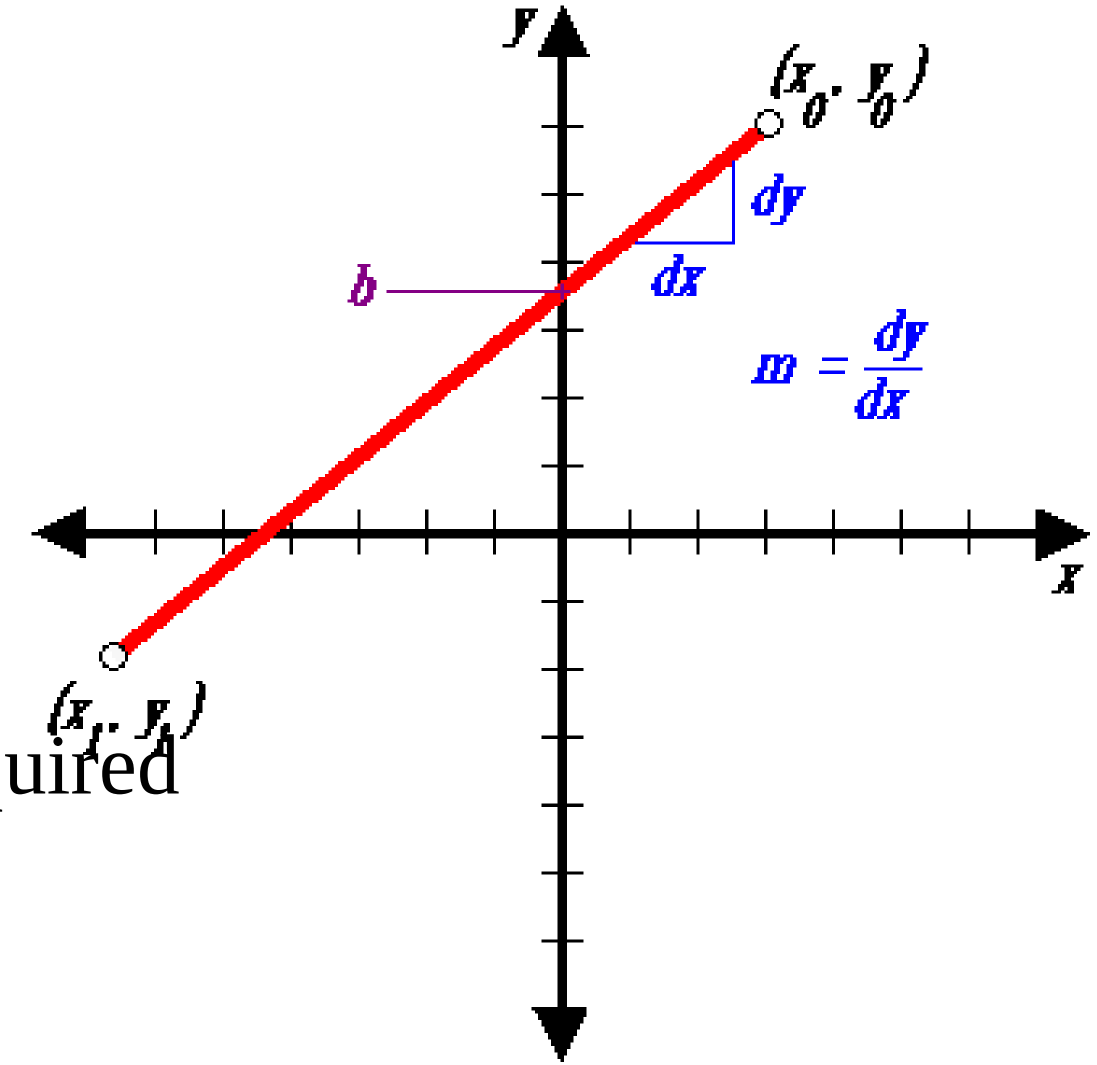
Based on *slope-intercept algorithm* from algebra:

$$y = mx + b$$

Simple approach:

increment x, solve for y

Floating point arithmetic required

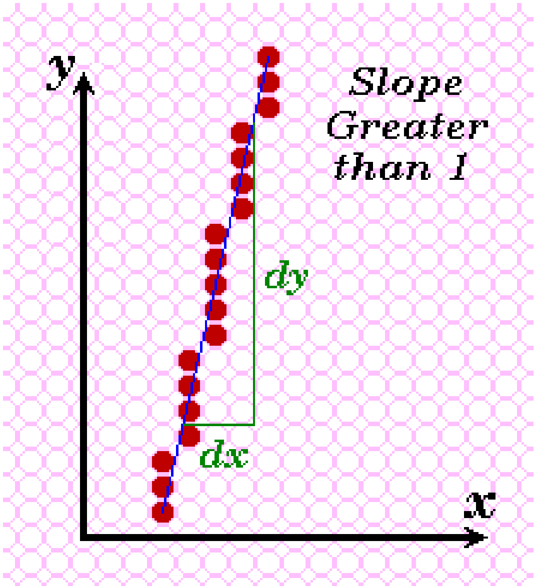


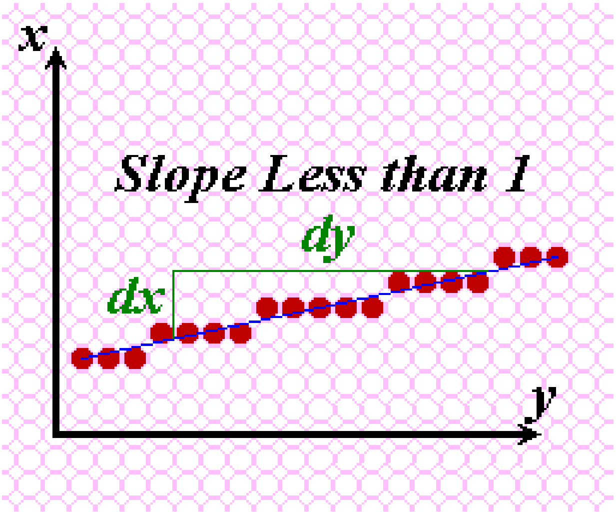
Does it Work?

It seems to work okay for lines with a slope of 1 or less,

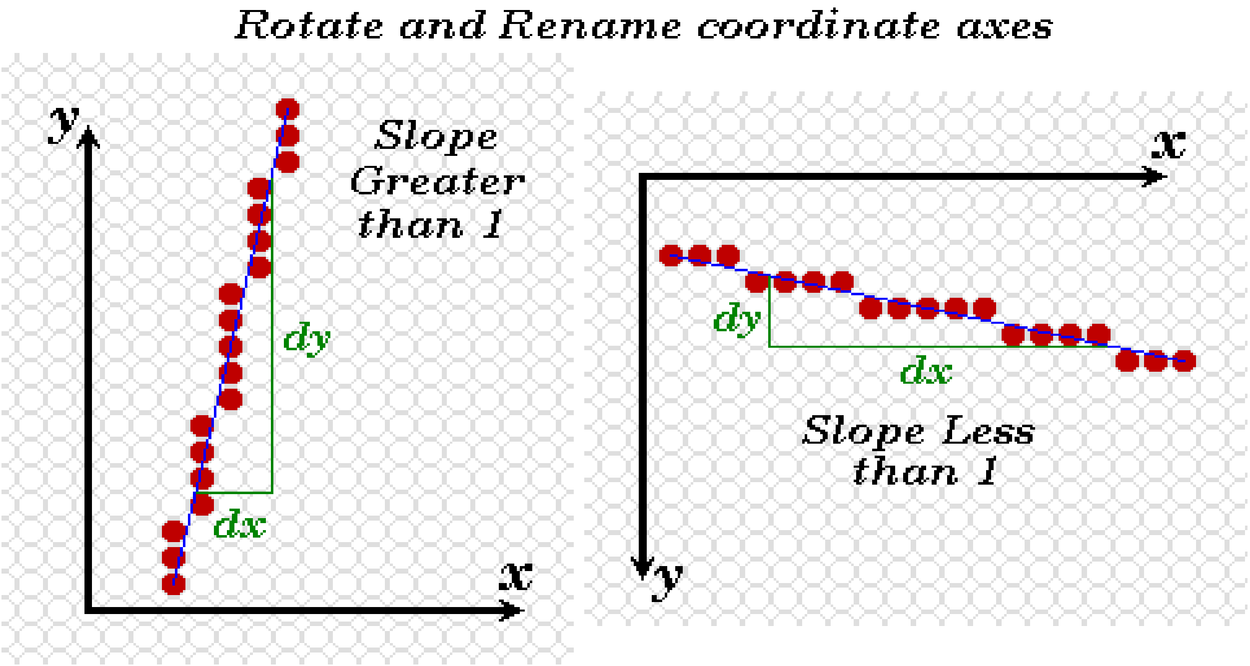
but doesn’t work well for lines with slope greater than 1 – lines become more discontinuous in appearance and we must add more than 1 pixel per column to make it work.

Solution? - use *symmetry.*





Modify algorithm per octant



OR, increment along x-axis if dy<dx else increment along y-axis

DDA algorithm

- DDA = Digital Differential Analyser
  - finite differences
- Treat line as parametric equation in t :

Start point - (x_1, y_1)
End point - (x_2, y_2)

$$x(t) = x_1 + t(x_2 - x_1)$$
$$y(t) = y_1 + t(y_2 - y_1)$$

DDA Algorithm

$$x(t) = x_1 + t(x_2 - x_1)$$
$$y(t) = y_1 + t(y_2 - y_1)$$

- Start at $t = 0$
- At each step, increment *t* by *dt*
- Choose appropriate value for *dt*
- Ensure no pixels are missed:
  - Implies: $\frac{dx}{dt} < 1$ and $\frac{dy}{dt} < 1$
- Set *dt* to maximum of *dx* and *dy*

$$x_{new} = x_{old} + \frac{dx}{dt}$$
$$y_{new} = y_{old} + \frac{dy}{dt}$$

$$dx = x_2 - x_1$$
$$dy = y_2 - y_1$$

DDA algorithm

```
line(int x1, int y1, int x2, int y2)

{
float x,y;
int dx = x2-x1, dy = y2-y1;
int n = max(abs(dx),abs(dy));
float dt = n, dxdt = dx/dt, dydt = dy/dt;
    x = x1;
    y = y1;
    while( n-- ) {
        point(round(x),round(y));
    x += dxdt;
    y += dydt;
    }
}
```

n - range of t.

DDA algorithm

- Still need a lot of floating point arithmetic.
  - 2 ‘round’s and 2 adds per pixel.

- Is there a simpler way ?
- Can we use only integer arithmetic ?
  - Easier to implement in hardware.

Summary of line drawing so far.

- Explicit form of line
  - Inefficient, difficult to control.
- Parametric form of line.
  - Express line in terms of parameter t
  - DDA algorithm
- Implicit form of line
  - Only need to test for ‘side’ of line.
  - Bresenham algorithm.
  - Can also draw circles.