Computer Graphics

**Chapter 1:**

**What is computer graphics?**

* Creation and manipulation of pictures using a computer

**Computer Generated Picture**: not a natural image

Uses of computer graphics:

* arts/entertainment
* Process monitoring
* Computer aided design
* Scientific analysis
* Virtual design

**Raster Image**: a collection of pixels, a pixel is the smallest drawable element with the colour as its attribute, image is stored in a two dimensional array(matrix)

**How are Raster images created?**

* Hand design images: designer must assign each pixel a colour,not practical
* \*Rendered images: The most powerful and difficult method. Images are created in steps. Images created by simpulation.\*
* Scanned Images: Real scene images digitized using a scanner

**Representation of colour in Raster images**

* **Bitmap Images**: colour has only 2 values; black(0) and white (1). Each pixel only requires 1 bit for storage.
* **Gray scale images**: Each pixel is represented by n bits(pixel depth). Black(0) to white (255). Each pixel requires one byte for storage. Ex. an 1000x1000 8-bit gray-scale image would require one million bytes
* **Colour Images**: Measures red, green and blue intensities. RGB model. Each set is an ordered triplet.The total number of bits used to store the colour of each pixel is often called **colour depth**. A colour depth of 3 in the RGB model provides 8 different colours (not realistic).

**True-colour**: the highest quality colour images , the colour depth is equal to 24

**The scanning Process:**

* can hold 64 colours
* Takes red,green and blue
* Shoots to a spot on the screen
* depends on x and y
* 3 beams of electrons hitting the small spot(RGB)
* 3 kinds of phospur
* Only 64 different colours
* 2 bits for each colour

**Picture Drawing:**

* Converts numeric values into actual voltage
* 3 electron beams hit a well defined physical location
* One dot glows red one glows blue and one glows green on the spot
* Because they are so close together they look as one to our eyes
* A picture is drawn row by row
* Order is from top to bottom and left to right
* Must be refreshed rapidly(60x per second) to avoid flicker

**Flat Panel Display:**

* Emissive: convert electric energy into light
  + LED
* Nonemessive: convert light from other sources into graphics patterns
  + LCD

**Plasma Display Panels(PDP):**

* Filled with gas(typically neon) wires are placed horizontally on one and vertically on the other. Voltage is applied to the intersection of the wires, which ignites the plasma

**Thin-Film electroluminescent:**

* Similar to pdp’s except the region is filled with phospur

**Light Emitting Diodes:**

* A matrix of diodes

**LCD’S:**

* Picture is obtained by passing polarized light through a liquid crystal material that can be aligned

**Colour Maps:**

* The most expensive frame buffers have a colour depth of 24 bits
* Image has a million pixels, max colour number is 256, saved in one million byte
* Ex.
  + A picture of 1000x1000
  + Using a 24 bit frame buffer, the storage area will be 3 million bytes, can display 16 million different colours at a time
  + Using an 8 bit frame buffer and an LUT of 24 bit entries, the storage area will be 1 million bytes for the picture and 768 bytes for the LUT, in this case we can display 16 million different colours any 256 at any time
  + Using a 12 bit frame buffer with an LUT of 24 bit entries the storage area will be 1.5 million bytes plus 12k bytes for the LUT, can display 16 million different colours, any 4000 at a time

**Computer graphics:** the goal is to created a picture, pictures are outputs

**Image Processing:** the goal is to process and extract info from natural  pictures, pictures are input

**Chapter 2:**

**Point plotting:** to plot a pixel at position (x,y) with the colour c, frame value is loaded with colour c

**Line Drawing:** accomplished by plotting all the pixels along the line between two given end point, coordinates are calculated from the equation of the line

**Line Drawing Algorithm:**

Equation of a line: ax + by + c = 0

Non-vertical line: y = mx + b

* Typically you are given two endpoints to draw the line
* M = y2-y1/x2-x1
* B = y1-m\*x1

**Digital Differential Analyser**

* From the previous equations, for any x interval, delta x the corresponding y interval delta y is given by;
* Delta y = m(delta x)
* Case of a Positive slope:
  + 0 ≤ m ≤ 1 : ∆x can be set to 1, and
    - ∆yk = yk+1 − yk = m∆x = m
  + That is, yk+1 = yk + m
  + m > 1 : ∆y can be set to 1, and xk+1 = xk + 1/m
* Case of a negative slope:
  + |m| ≤ 1 : ∆x can be set to 1, and yk+1 = yk + m
  + |m| > 1 : ∆y can be set to 1, and xk+1 = xk + 1/m

**Bresenham's Line Algorithm**

* Uses only incremental integer calculations
* Can be adapted for circles and other curves
* Consider the case of positive slope
  + Goal of this algorithm is to find a decision parameter not requiring multiplications
* Consider the positive case
  + 1. Read left and right segment endpoints, (x0, y0) and (x1, y1)
  + 2. Calculate the constants : ∆x, ∆y, 2∆y, 2∆y − 2∆x and pk = 2∆y − ∆x
  + 3. if (m > 0) then step = 1 else step = -1
  + 4. set x = x0 and y = y0
  + 5. While (x <= x1)
  + • Draw pixel at (x, y)
  + • x = x + 1
  + • if (pk < 0) then pk = pk + 2∆y else y = y + step pk = pk + 2∆y − 2∆x

* 1/R to calculate pixels in a circle, 1 pixel l

**Other cases of slope m:**

* • ∆x = 0 : vertical line, draw it using a simple loop
* • ∆y = 0 : horizontal line, draw it using a simple loop
* • |m| = 1 : special case, draw it using a simple loop
* • |m| > 1 : Interchange the roles of x and y directions. That is, we step along the y-direction in unit steps and calculate succesive x values.

**Circles:**

* Center position (xc,yc)
* Radius r
* The set of points in a circle must satisfy
  + (x − xc) ^2 + (y − yc) ^2 = r^ 2
* We can generate all pixels of a circle by calculating only the pixels within the sector : x = 0 to x = y. This arc represents only 1/8 of the total pixels.

**Filled Area Primitives:**

* Finding the overlap for scan lines crossing the area
* Starting from a given inward position the paint outward until the boundary is encountered

**Character Generation**

* Design style for a character type is called typeface, also fonts
* Bitmap font: a two dimensional bit face. Simple yet requires more space.
* Outline font: characters are described using straight lines and curves. More time yet less space.

**Attributes of Output primitives**

* Any parameter that affects the way a primitive is to be displayed is an attribute parameter
* Colour, size

**Chapter 3:**

**Translation:**

* A geometric transformation that moves an object along a straight line path from one location to another
* A point is translated by adding the vector coordinates of translation.
* Ex.

* A translation is a rigid transformation
* To apply a translation to a circle
  + translate the center to (xc + tx, yc + ty)
  + Redraw the same circle at the new location

**Rotation**

* A geometric transformation that moves an object along a circular path from one location to the next
* A rotation angle specifies how much rotation is to be performed
* A rotation point specifies the point around which the object is to be rotated
* A rotation can be described by a matrix but a matrix is not a rotation, to be a rotation is must be an identity matrix when rotated with its smaller matrix?
* x θ = x cos θ − y sin θ
* y θ = x sin θ + y cos θ
* Matrix form

* Compact form of the above is P’=RP
* Where R is a 2x2 orthonormal matrix RR^T=I

* The previous rotation equations can can be generalized by including the coordinates of the rotation axis.
  + x’ = xr + (x − xr) cos θ − (y − yr) sin θ
  + y’ = yr + (x − xr) sin θ + (y − yr) cos θ
* Rotations are rigid transformations

**Scaling**

* Alters the size of an object
* Fixed point: location a scaled object can be controlled by a special point
  + x’ = sxX + xf (1 − sx)
  + y’ = syY + yf (1 − sy)
* x’ = sxX and y’ = syY
* Can be written in matrix form:
* P’ = SP
* Uniform scaling: obtained when Sx = Sy

**Matrix Representation and Homogeneous Coordinates**

* P’ = M1 · P + M2
  + Where M1 and M2 represent the multiplicative and additive factors respectively.
  + Matrix form:

* However, in general we can have any value for the last row in the point coordinates. After adding a third component, the cartesian coordinates (x, y) becomes (hx, hy, h) when h is any nonzero value. These coordinates are called homogeneous coordinates.

**Composite Transformations:**

* Translations:

* Rotations:

* Scaling:

**General Rotations:**

A rotation around an arbitrary axis can be achieved by the following three steps:

1. Translate the object such that the axis position moves to the origin.

2. Rotate the object around the origin.

3. Translate the object such that the axis position moves back to its original position.

* Using matrix operations, the composite transformation for this sequence is given by
  + T(xr, yr) · R(θ) · T(−xr, −yr)
* Matrix:

**General Fixed Point Matrix**

1. Translate the object such that the fixed point moves to the origin,

2. Scale the object,

3. Translate the object such that the fixed point moves back to its original position.

* Using matrix operations, the composite transformation for this sequence is given by
  + T(xf , yf ) · S(sx, sy) · T(−xf , −yf )
* Matrix:

**Composite Transformation Properties**

* Matrix product is associative
  + A · B · C = (A · B) · C = A · (B · C)
* Matrix product is noncommutative
  + A · B != B · A
* This is a very important issue. For instance, if an object is to be translated then rotated, the order must be observed.
* Pure rotations is fine, second after the first would be the same

**Form of General Transformation**

* Translation, rotations and scaling can all be represented in the same matrix
* Last row is to accommodate the homogenous, put one in the last spot and you know it is 2d
* M = T(tx, ty) · R(xf , yf , θ) · S(xf , yf , sx, sy)
* Matrix:

**Transformations Between Coordinate Systems**

* It is common that an object in one coordinate system must be transformed into another coordinate system
* A scene is described is a coordinate system, call it scene-coordinates
* An object, described in a different coordinate system, call it object-coordinates.
* A rotation plus a translation
* Superimpose the object coordinate system onto the scene coordinate system
* A translation to bring the origin (x0, y0) of the object coordinate system on the origin of the scene coordinate system

* A rotation around the scene origin to superimpose the OX-axis on the X-axis.

* The object-to-scene transformation matrix MO2S is given by
  + MO2S = R(−θ) · T(−x0, −y0)

**Affine Transformations**

* An affine transformation is a linear transformation where p 0 = A · p

* The above suggest that any 3 × 3 matrix with the last row defined by (0, 0, 1) is an affine transformation.
* Parallelism : two parallel lines will remain parallel after an affine transformation
* Ratios : a ratio between two lengths will remain the same. For instance, if a point m was the middle points of p and q, then the transformed point m0 will remain the middle point of p 0 and q 0 .
* Finite points will remain finite points.
* An affine transformation does not necessarily preserve the shape

**Transformation Function in OpenGL:**

* glRotatef(angle, x, y, z) : where,
  + Angle, in degrees, is the rotation angle
  + (x, y, z), a vector, are the 3D coordinates of the rotation axis.
* glTranslatef(x, y, z) : where, (x, y, z) is the translation vector
* glScalef(sx, sy, sz) : where, sx, sy and sz are the three scale factors along the coordinate axis.