Introduction to Graphics Revision Notes

Kevalee Shah

April 28, 2019

Contents

1 Introduction

1.1 Definitions

- Image - 2D function, the value at any point is an intensity or colour. Not digital

- Digital Image - not actually digital, but instead it is a sampled and quantised version of a real image. rectangular array of intensity values
- Pixel - each value in a digital image

  measured in ppi or dpi - pixels/ dots per inch
- Frame Buffer - piece of memory reserved for the storage of the current image being displayed.

1.2 Human Eye

- Retina - array of light detection cells, found at the back of the eyes
- There are two types of light detectors - rods and cones.
- Cones - three types: short, medium and long. They allow us to see in colour
- Macula - region of the retina densely packed with cones
- Fovea - high resolution area of the retina, highest density of cones, therefore the highest resolution vision, found in the centre of the macula
- Optic Nerves - take signals from the retina to the visual cortex in the brain
- The lens forms an image on the retina

1.3 Colour Quantisation

- Process that reduces the number of distinct colours used in an image, ensuring that the new image is as similar to the original image.
- The intensity values need to be quantised for digital storage. This means the brightest intensity that can be stored in limited.
- The range is between 8-16 bits
- Colour is stored as a set of numbers - usually 3 numbers each between 5 - 16 bits
- 8 bits was the standard for greyyscale images, for the intensity. 8 bits is one byte. Images of size $W \times H$ is stored in a block of $W \times H$ bytes. Memory is 1D, but images are 2D and therefore to store the pixels, `pixel[x][y]` is stored at $base + x + (W \times y)$
- Colour values used to be 24 bits per pixel - one byte each for red, green and blue. It is more common to store each colour in its own plane - 3 lots of $W \times H$ values.

2 Ray Tracing

2.1 Overview

Rendering technique that is used to generate an image by tracing the path of light. The computer sends ray to every pixel in the scene. When a ray hits an object, the computer sends more rays to the light sources in the scene. The light source affects the colouring and brightness of the pixel. Ray tracing allows for reflections and refractions. Shadows can be made by seeing if the reflected ray meets another object on its way to the light source and. We know that means there is another object blocking the light, meaning there must be a shadow. Reflexive objects send out another ray at a certain angle in order to check if there are other objects which should be reflected at the intersection point.

2.2 Algorithm

```
select an eye point and a screen plane
FOR every pixel in the screen plane
   determine ray from eye to the pixel centre
   FOR every object in the scene
      IF the object is intersected by the ray
         IF the intersection is the closest so far to the eye
            record intersection point
   set the pixel colour to that of the object at the closest
       intersection point
```

2.3 Effects

Shading Once you have the intersection of a ray with the nearest object, the normal of the object at the intersection point can be determined. Shoot rays from the point of intersection to all of the light sources. The diffuse and specular reflections of the objects can be calculated. The combination of diffuse, specular and ambient components determines the colour.

Shadows When tracing the ray back to the light sources, if it meets another object then there must be a shadow. However be aware of self shadowing.

Reflection If surfaces are reflective then rays can be spawned to find the contribution to the pixel's colour given by the reflection.
Specular reflection has interesting properties due to the occulsion of micro-facets by one another.
Type of reflection:

- Perfect specular - perfect reflection
- Imperfect specular - a few scattered rays around the main reflected ray
- Diffuse - many scattered rays all around the main incoming ray

Transparency Objects can be totally or partially transparent. This means that objects behind can be seen. Transparent objects have refractive indices, which determine how much the ray bends.

2.4 Shading Assumptions

- Only diffuse shading
- All light falling on an objeect comes directly from a light source - no interaction between objects
- No objects cast shadows on eachother - can think of each surface as the only one in the scene
- Light sources are considered to be infinitely distant from the object, and therefore the vector to the light is the same across the whole surface.

2.5 Diffuse Shading

Equation:

$$\begin{aligned} I &= I_l k_d \cos\theta \\ &= I_l k_d (n \cdot L) \end{aligned}$$

This equation calculates the intensity of the light reflected by the surface.

- I_l - intensity of the light source
- k_d - proportion of light which is diffusely reflected by surface
- N - normal to the surface
- L - normalised vector pointing to the light source
- θ - angle between N and L

2.6 Specular Shading

This is an approximation to specular shading, by Phong.
Equation:

$$\begin{aligned} I &= I_l k_s \cos^n \alpha \\ &= I_l k_s (R \cdot V)^n \end{aligned}$$

- L - vector pointing in the direction of the light source
- R - vector of perfect reflection
- N - normal to the surface
- V - normalised vector pointing to the viewer
- I_l - intensity of the light source
- k_s - proportion of loght which is specularly reflected by the surface
- n - Phong's roughness coefficient. As n increases the specular reflection decreases.
- I - intensity of the specularly reflected light

2.7 Ambient Illumination

With the shading assumptions we assumed that there was no interaction between surfaces. In order to overcome this assumption, we can assume that all light reflected off all other surfaces can be amalgamated into a constant term - the ambient illumination.

Overall shading equation: $I = I_a k_a + \sum_i I_i K_d(L_i \cdot N) + \sum_i I_i K_s(R_i \cdot V)^n$

2.8 Sampling

Assumption: The value of the pixel is the colour of the object which lies exactly in the centre of the pixel.

Problems: This leads to jagged edges of objects, small objects missed completely, or thin objects being missed or split up into small pieces. These anomalies are known as *artefacts* and all the unwanted effects in an image are known as *aliasing*

2.9 Anti-aliasing

Methods to overcome the effects of aliasing are known as anti-aliasing.

2.10 Ray-tracing Sampling

- Single point - through the centre of the pixel
- Super sampling - multiple rays through each pixel and then averaged

  Regular grid - divide pixel into sub-pixels and then centre of each of the smaller squares, if the size of the sub-pixel isnt small enough then the same issues as above can arise

  Random - N rays shot at random points in the pixel. Due to random nature may still lead to aliasing issues, as some samples might end up being unecessary and some areas would be lacking

  Poisson - N rays shooted at random points in the pixel, provided that the points are at least ϵ apart. Hard to implement, but best method of anti-aliasing.

  Jittered - Divide pixel into N subpixels. Shoot a ray into each pixel, at random. An approximation to Poisson, but easier to implement.

2.11 Distributed Ray Tracing

Refinement of ray tracing to allow for the rendering of softer effects. Normally single rays are used with only one reflection and transmission ray leading to sharp shadows and reflections, which is often unrealistic. Softer shadows, blurry reflections, blurry transmissions can be achieved by averaging multiple rays distributed over an area.

- Distribiting samples over the area of the pixel - Super sampling, described above, used for anti-aliasing
- Distributing samples over the light source area - means that light doesn't need to be treated as a single point and produces the effect of soft shadows.
- Distributing camera position over some area - allows simulation of camera with apeture. This produces depth of field effects
- Distributing samples in time - motion blur images can be produced for moving objects.

2.12 Maths

To check if a ray intersects with an object:

1. Write the vector equation of a ray: $\mathbf{P} = \mathbf{o} + s\mathsf{d}$
2. Write the equation of the object also in vector form.
   e.g.

   Sphere: $(\mathbf{q} - \mathbf{c}) \cdot (\mathbf{q} - \mathbf{c}) - r^2 = 0$

   Plane: $\mathbf{P} \cdot \mathbf{N} + d = 0$

   Right circular cone, vertex at the origin:
   $(\mathbf{u} \cdot \mathbf{d}) - |d|\,|u| \cos\theta$
   (u - the point, d - parallel to axis, θ - half the apeture)
3. Substitue the equation of the ray in the object. Solve for s
   e.g.

   Sphere: $((\mathbf{o} - \mathbf{c}) + s\mathsf{d}) \cdot ((\mathbf{o} - \mathbf{c}) + s\mathsf{d}) - r^2 = 0$, this leads to a quadratic equation in s. Need to check which solution is closer.

2.13 Advantages and Disadvantages

Advantages

- Realistic simulation of lighting compared to other methods
- Easier to have effects such as shadows and reflections
- Able to parallelise as each ray is independent
- Very high visual quality

Disadvantages

- High performance (around the square of object number) and therefore very slow - therefore not suitable for real-time applications such as video games
- Still doesn't produce completely photorealistic images

3 Rasterization

Overview Rasterization is the process of taking vector data and transforming it into pixels. 3D models are made up of (x, y, z) coordinates. However screens are 2D and therefore the rasterizer takes every polygon and calculates where to put it on the screen. The shape is made into a group of pixels.

3.1 3D Objects into Polygons

Polyhedral surfaces, including curved surfaces are made up from meshes of polygons. They are mostly made up of triangles, as 3 vertices must be planar - there is no ambiguity.
GPUs are optimised to split polygons into triangles.

3.2 Transformations

3.2.1 2D

- Scale: $\begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix}$
- Rotate: $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
- Shear parallel to x $\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$
- Shear parallel to y $\begin{bmatrix} 1 & 0 \\ a & 1 \end{bmatrix}$

An issue is that translations cannot be represented with 2×2matrices.

3.2.2 Homogeneous co-ordinates

- This form of coordinates allows for translations to be represented as well.
- $(x, y, w) \equiv (\frac{x}{w}, \frac{y}{w})$
- When $w = 0$, the point is at infinity.
- The above tranformations are the same but with an added row and column, e.g. the following matrix represents a shear transformation

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translations

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

This gives:

- $x' = x + wx_0$

- $y'y + wy_0$
- $w' = w$

In normal coordinates this is:

- $\frac{x'}{w'} = \frac{x}{w} + x_0$
- $\frac{y'}{w'} = \frac{y}{w} + y_0$

Note:

- The order of transformations is the reverse of how to right it down. e.g. shear then rotate is $[rotate] \times [shear] \times [point]$
- 3D homogeneous: $(x, y, z, w) = (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$
- 3D x-axis rotation:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3 Projection

This is the process of mapping a 3D world onto a 2D plane

Parallel This is an unrealistic way of projecting:

$$(x, y, z) \to (x, y)$$
$$(x, y, z) \to (x, z)$$
$$(x, y, z) \to (y, z)$$

Perspective This is when things get smaller the farther back they get to give a more realisitc image.
We want

$$x' = x\frac{d}{z}$$
$$y' = y\frac{d}{z}z' = \frac{1}{z}$$

The matrix needed to get this is

$$\begin{bmatrix} x \\ y \\ \frac{1}{d} \\ \frac{z}{d} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The assumptions are the the screen is centred at $(0, 0, d)$, parallel to the $x - y$ plane, the z axis is on the plane and the eye is at the origin. For arbitary points it is easier to transform all objects into standard viewing coordinates and then use the above assumptions.

3.4 Sequences of Transformations

1. Object in object coordinates
   $\xrightarrow[\text{transform}]{\text{modelling}}$
2. Object in world coordiantes
   $\xrightarrow[\text{transform}]{\text{viewing}}$
3. Object in viewing coordinates
   $\xrightarrow[\text{projection}]{}$
4. Object in 2D screen coordinates

Matrices

- Model Matrix - positions each object in the scene - could be different for each object
- View Matrix - position all objects relative to the camera
- Projection Matrix - 3D coordinates are projected onto a 2D plane (keep z coordinate for depth testing)
- Together:
$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = P \cdot V \cdot M \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Viewing Transformation In order to transform an arbitary coordinate system to the default viewing system as series of transformations need to be done. In the world coordinates

- camera at $e_x, e_y.e_z$ (e)
- look point (centre of screen) at l_x, l_y, l_z (l)
- (u) - perpendicular to **el**

Transformations:

- Translate eye point to origin (**T**)
- Scale so that eye point to look point distance is the distacne from the origin to the screen centre (**S**)
- To align **el** with the $z-$axis:

  Transform e and l to the new coordinate system
$$e'' = S \times T \times e = 0 \qquad l'' = S \times T \times l$$

  Rotate **e''l''** into $y - z$ plane, by rotating about the $y-$axis.

  Now rotate the viewing vector along the $x-$axis so that it aligns with the $z-$axis
$$l''' = R_1 \times l''$$

- Check that up vector points up - rotate the up vector about the $z - axis$

$$u'''' = R_2 \times R_1 \times u$$

Overall this is

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} R_3 \times R_2 \times R_1 \times S \times T \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$e \rightarrow (0, 0, 0)$
$l \rightarrow (0, 0, d)$

Transformation of Normal Vectors Transformation by a non-orthogonal matrix does not preserve angles, and therefore the normal changes

Suppose N is the normal to the vector T on a surface.

$$N \cdot T = 0$$
$$N' \cdot T' = (GN) \cdot (MT) = 0$$

G is the normal transformation, M is the vertex transformation

$$G = (M^{-1})^T$$