CPSC 453

David Ng

Fall 2016

Contents

1 September 12, 2016

1.1 Topics Covered in Class

1. Light and Colour
2. Digital Imaging
3. Geometry
4. Textures and Materials
5. Graphics Systems

2 September 14, 2016

2.1 Computer Graphics

Computer graphics is the discipline concerned with generating or manipulating visual imagery using computational devices and methods.

Remark. The **wagon wheel effect** refers to the effect seen when it appears that wheels rotate backwards in video due to the frame rate. (This relates to the problem of sampling).

2.2 Light, Colour, and Displays

Light is an electromagnetic wave, as first described by James Clark Maxwell. We note that visible light is only a tiny part of the electromagnetic spectrum, including radio waves, microwaves, gamma rays, ultraviolet rays, etc. Black-body radiation refers to an opaque, non-reflective body in thermodynamic equilibrium that emits electromagnetic radiation with a spectrum dependent only on its temperature.

The temperature settings affect the "blueness" that the white point of the camera is set to. Thus, it calibrates what the camera or display considers to be white. Colours however, are subjective to a certain degree. In terms of human colour perception, we have two types of photoreceptors called rods and cones in our retinas. We have three types of cones that are used primarily for bright situations, whereas rods are used in low light situation.

Luminous efficiency results as humans are not equally sensitive to all wavelengths of light. Light at long wavelengths is primarily detected by the L-cones, peaking at around 560 nm, the medium wavelengths peaking at around 530 nm is detected by the M-cones and the short wavelengths are detected by the short S-cones. Humans have trouble distinguishing colours of blue, as our cones are less sensitive to that wavelength. Since colours are in the eye of the beholder, our perceptual systems may be tricked and thus may not necessarily represent reality accurately.

3 September 16, 2016

3.1 Graphic Displays

Graphics displays include not only televisions, but also printed material as well. In graphic displays, our goal is to stimulate photoreceptors in some combination. When choosing primary colours to stimulate photoreceptors, we note that choosing wavelengths close together make it so that there is a difficulty in generating other colours. A limitation in the traditional RBG method is that red is at a wavelength near the end of the spectrum that is largely undetected by the L-cone. In practice, the red wavelength is shifted towards the others, making it more receptive by the L-cone at the cost of some forms of red.

3.2 Additive Color Blending

We add the colours red, green, and blue to create the other colours. CIE Chromaticity Diagrams give us an understanding of the range of colours possible by a choice of three primary colours. In colour matching, we note that we can achieve a certain colour by using a linear combination of the three primary colours. However, some colours require "negative" red light. In the Standard RGB Gamut, we take the three primary colours to be red, green, and blue. This creates a colour space determined by the triangle formed by the wavelengths of these colours in the CIE Diagram.

White light can be refracted to view its component colours. The light emitted by light bulbs appear to emit white light. However, when a CD is used to reflect the light, it can be seen that there are certain distinct colours, and is thus the light is not perfectly white in the scientific sense. Cyan, magenta, and yellow can be chosen as subtractive colors. Halftones are used to change the intensity of the colours when printed. We note that the choice of primary colours is arbitrary. It is not possible to choose three colours such that the entire spectrum is visible. However, a good choice of primary colours presents a close approximation.

3.3 Resolution

The human foveola is the point where visual acuity is sharpest. The highest visual resolution is about one arc minute (1/60 of a degree). Thus, a phone does and does not have the same resolution as high definition TV. It depends on how close or far they are used.

A digital image is simply an encoding of bits and bytes. We note that there are generally two key elements that comprise a digital image. They are colour and resolution. Colour can be expressed as RGB, or CMYK. Resolution is a sample of "pixels" arranged in a two dimensional grid. They may or may not correspond to pixels on the display. It is important to remember that displays are engineered to human visual perception. Resolution is not necessarily measured in pixels, and digital images are the goal of computer graphics.

4 September 19, 2016 - September 21, 2016

4.1 Geometric Representation

To represent a circle in binary representation, we could store information on pixels, centre/radius, and pi. Our goal to generate digital images, requires us to draw squares, circles, and other shapes into a pixel grid. We note that there are four geometric representations of images:

1. **Discrete**: We can take samples along the curve and store that information. Then connect those points with lines to provide an approximation of the curve. We note that with just points and lines, we cannot accurately represent a circle. For a square, we simply need four points and four lines exactly to represent a square. OpenGL ultimately reduces to describing points, lines, and triangles. Discrete samples are easy to use and expressive, but are often only approximations and require many points.

2. **Implicit**: In explicit form, we have one of the variables written in terms of another. For instance, a parabola could be represented as

   $$y = x^2.$$

   In implicit form, the implicit relation would be expressed as

   $$x^2 - y = 0.$$

   Conic sections are of the form

   $$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0.$$

   We note that this form can perfectly describe a circle, but we are no longer able to express a square with finite exponents. While the ability to describe functions and store coefficients makes this form of representation compact and intuitive, it is also difficult to render.

3. **Parametric**: Parametric equations express points on a curve or surface as a function of a free parameter. For instance, we have the parametric equation

   $$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sin(u) \\ \cos(2u) \end{bmatrix}$$

   which produces an infinity shape when plotted. While parametric equations are fairly convenient to render, the representation is often not unique and the shape may not be immediately intuitive.

4. **Generative functions**: These functions include fractals, L-systems, subdivision schemes, textures, noise, and different types of terrain. While they map very well to code, they excel at describing only a narrow set of phenomena.

It is important to remember that each representation has its own advantages and disadvantages.

5 September 23, 2016

5.1 Graphics Systems

Early graphics systems included Sketchpad, a computer program written by Ivan Sutherland in 1963, plotters such as the ZUSE Graphomat Z64, and cathode ray tubes. Raster scanning to produce images on a cathode ray tube involved many component parts, including the deflection voltage electrode, the electron gun, the electron beam, the focusing coil, and a phosphor-coated screen. Bit-mapped required the CRT electron beam scan out the contents of a memory buffer, with 0 indicating that the bit was off, while 1 meant that it was on. 3D rendering became possible as advances in the field made possible scan line conversion, texture mapping, compositing, and z-buffering.

5.2 Graphics Pipeline

A graphics pipeline transforms data into images. The primitives that are used in OpenGL include points, lines, and triangles. Rasterization and scan conversion of these primitives is taken care of, and deal with the images that is shown when a certain point, line, or triangle is geometrically specified. Using Bresenham's algorithm and other techniques, OpenGL performs scan line conversion, image sampling, and interpolation. The OpenGL pipeline includes the stages of vertex specification, vertex shading, tessellation, geometry shading, rasterization, and fragment shading. Almost all of these stages is programmable. We simply need to specify inputs and outputs at each stage, while OpenGL handles the compiling and running of the code on the GPU.

6 September 26, 2016

6.1 Interpolation

Interpolation is defined as the process of fining "in-between" values. The basic form of the parametric equation for interpolation between points A and B is given as

$$R = (1-t)A + tB, \quad t \in [0,1]$$

We note that the above equation is possibly the most used function in computer graphics. It is used to convert a discrete function, $f(x) : \mathbb{Z} \mapsto \mathbb{R}$, to a continuous function, $g(x) : \mathbb{R} \mapsto \mathbb{R}$.

6.2 Use of Interpolation

We can utilize interpolation for the following applications:

- Vectors and Points (although we should not be adding points).
- Lines and Triangles.
- Angle and Joint Articulations.
- Orientations (flight paths) and Camera Angles
- Colours

We can also interpolate other features. For instance, displacement vectors, image pixels, and time progression can all be interpolated.

6.3 Two-Dimensional Interpolation

We can convert a discrete 2D function, $f(x, y) : \mathbb{Z}^2 \mapsto \mathbb{R}^2$, to a continuous one, $g(x, y) : \mathbb{R}^2 \mapsto \mathbb{R}^2$. An example of when we would like to apply two-dimensional interpolation is for image manipulation. **Bilinear interpolation** can be used to compute three distinct linear interpolations to arrive at the desired result in two dimensions. We can also consider bilinear interpolation as a height function. The surface that arrises from the interpolated continuous function form is a ruled surface. That is, it takes the form of a hyperboloid. As opposed to nearest neighbour, where each square group of pixels adopts a discrete colour distinct from the other groups around it, bilinear interpolation smooths these differences out through its interpolated result.

7 September 28, 2016

7.1 Sampling

Sampling is the recording of a continuous signal at discrete points. We use sampling to convert a continuous function to a discrete one. To obtain the original function back, we just use interpolation. According to the **Shannon Sampling Theorem (Nyquist Frequency)**, to faithfully reconstruct a band-limited signal from its samples, the sampling rate must be at least twice that of its highest frequency component. Jean-Baptiste Fourier additional claims that every continuous and periodic function may be expressed as the sum of a series of sine or cosine functions, each with specific amplitude and phase. **Aliasing** is the result of undersampling, and produces a seemingly lower frequency during a reconstruction of the original signal.

7.2 Digital Audio and Music

One of the first and most prevalent uses of digital sampling and signal reconstruction is for digital audio and music. An analog to digital converter allows one to store sound. A digital to analog converter is then used to allow one to recreate and play the sound. The typical range of audible frequencies is between 20 Hz - 20 kHz. The typical sampling rates for digital audio recordings are from 44.1 kHz - 96 kHz, which we note is just slightly more than twice the typical range of audible frequencies.

To remove aliasing, we make use of a **low-pass filter**, which is a resistor and capacitor that removes high frequency. That is, it cuts off parts of the signal that possess a high frequency in order to smooth out the signal. This results in a band-limited signal. For temporal signals, we have a signal repeating after a certain amount of time. Spatially, we can relate temporal frequency for a spatial analogue, with the only difference being the units in which frequency is measured.

7.3 Frequency Content of Shapes

A simple shape such as a square or a box is not a band-limited signal. Fourier noted that we can decompose it, but it has an infinite spectrum. So how do we employ anti-aliasing for two-dimensional images? According to the Nyquist sampling theorem, downsampling to a smaller image from a higher-resolution original produces aliasing artifacts unless the downsampling occurs after applying an anti-aliasing filter. If edges of high frequency appear near each other, aliasing appears as a moire pattern. This occurs because the sampling theorem's conditions are not satisfied when the image is rescaled to a smaller size. Since the Gaussian blur has the effect of reducing an image's high frequency components, it is a low pass filter and can therefore be used for anti-aliasing purposes. Thus, when we apply the Gaussian blurs, it counteracts the effects of aliasing.

8 September 30, 2016

8.1 Convolution and Digital Image Effects

Convolution is an operation on two functions, and is denoted by

$$f(x) \star g(x) \implies (f \star g)(x).$$

For instance, consider the continuous and discrete forms of the moving average:

$$h(x) = \frac{1}{2r} \int_{x-r}^{x+r} f(t)\mathrm{d}t, \quad f(x), x \in \mathbb{R}$$

$$c[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} a[j], \quad a[i], i \in \mathbb{Z}$$

Convolution would therefore be a weighted moving average about another function:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)\mathrm{d}t$$

$$(a \star b)[i] = \sum_{j} a[j]b[i-j]$$

Most convolution kernels in the real world have finite support. That is, the second function is zero outside of a certain radius. We note that convolution is commutative, associative, distributive (over addition), and has an identity element. Discrete-continuous convolution is also possible, and is employed in signal reconstruction to reproduce a continuous signal by convolution with a reconstruction kernel such as a cubic spline.

8.2 Convolution in 2D

If we have functions in two or more variables, we can similarly define 2D convolution. For instance, for discrete functions $a[i,j], b[i,j] : \mathbb{Z}^2 \mapsto \mathbb{R}$, the 2D convolution is defined as

$$(a \star b)[i,j] = \sum_{i'} \sum_{j'} a[i',j']n[i-i', j-j']$$

This may be applied to sampled digital images, which can be described as a discrete function of two variables. Different effects can be achieved, such as edge detection and blurring. Edge detection in 2D is performed by applying either the vertical or horizontal Sobel filters, defined respectively as

$$G_x[i,j] = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

$$G_y[i,j] = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Remark. To apply these filters, we flip the kernels along the middle vertically and horizontally. Then, for each 3x3 block of pixels in an image to which the filter is applied, we multiply the value of each pixel by the corresponding entry of the kernel, then sum the result. The pixel which was positioned under the middle entry of the kernel adopts the new computed value. This is repeated for each pixel in the original image, with edges often handled by treating the pixel value as 0.

9 October 3, 2016

9.1 Blurring

We can use the $2D$ Gaussian Function for blurring purposes. Given that σ denotes the standard deviation, note that it is given as

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

To evaluate for discrete functions, we simply apply the function to discrete points. This gives the discrete convolution kernel

$$G[i,j] = \begin{bmatrix} 0.04 & 0.12 & 0.04 \\ 0.12 & 0.36 & 0.12 \\ 0.04 & 0.12 & 0.04 \end{bmatrix}.$$

We may choose to use it for censoring, anonymity, effects, etc. It may also be used for anti-aliasing.

9.2 Signal Reconstruction

Most interpolations can be expressed as convolutions. It is not always the case that convolution is interpolation. The box function, when used as a convolution kernel, provides "nearest neighbor" convolution. The tent function provides linear convolution. In two-dimensions, these functions can also be used to provide convolution. To faithfully reconstruct the original signal, we use the *sinc* function:

$$\operatorname{sinc}(x) = \frac{\sin(x)}{x}$$

The Fourier transform takes a function and expresses it as a sum of cosines. The sinc function is the Fourier transform of the box function. The Fourier transform of the tent function is the sinc^2 function. We also note that the box function convolved with itself is the tent function, while the tent function convoluted with itself is equal to a function which when Fourier transformed, gives the sinc^4 function.

We note that the Fourier Transform and its inverse are symmetric. For the perfect reconstruction of a band-limited signal, we simply apply convolution with the sinc function. However, due to the high computational cost of evaluating the sinc function, other reconstruction kernels are often used in practice. Signal reconstruction is performed by convolution with a reconstruction kernel. Aliasing is a result of leftover artifacts, possibly as a result of poor reconstruction.

9.3 Colour Conversion

A sampled colour image maps integer pairs to three colour channels

$$I[i,j] : \mathbb{Z}^2 \mapsto \mathbb{R}^3$$

We consider each colour as a separate function to apply filters

$$R[i,j] : \mathbb{Z}^2 \mapsto \mathbb{R}$$
$$G[i,j] : \mathbb{Z}^2 \mapsto \mathbb{R}$$
$$B[i,j] : \mathbb{Z}^2 \mapsto \mathbb{R}$$

This allows us to separate the red, green, and blue colour channels of an image. We may also choose to average the channels to obtain a greyscale image. There are average and perceptual methods for conversion to greyscale. Simply taking the average colours to get a greyscale image may not account for human perception as a result of luminous efficiency. .

9.4 Review of Vectors

Definition. A **vector** is a quantity with a magnitude and a direction, where the magnitude is a real number.

10 October 5, 2016

10.1 Vectors Review

We reviewed basic vectors and vector operations.

11 October 7, 2016

11.1 Vectors Review

We recall that the **scalar product** is defined as

$$\vec{a} \cdot \vec{b} = \|\vec{a}\|\|\vec{b}\| \cos(\theta).$$

The **vector product** is defined as

$$\vec{a} \times \vec{b} = \|\vec{a}\|\|\vec{b}\| \sin(\theta) u.$$

We note the following properties

- $$\vec{a} \times \vec{b} = -\left(\vec{b} \times \vec{a}\right)$$
- $$\left(\vec{a} \times \vec{b}\right) \times \vec{c} = \vec{a} \times \left(\vec{b} \times \vec{c}\right)$$
- $$\left(\vec{a} + \vec{b}\right) \times \vec{c} = \vec{a} \times \vec{c} + \vec{b} \times \vec{c}$$

Definition. A **vector basis** is a set of linearly independent vectors that span a space. An **orthonormal basis** is a set of unit vectors, all orthogonal to one another, which form a basis.

11.2 Basis Conversion

Let $\vec{v} = v_1\vec{a_x} + v_2\vec{a_y}$, where $v_1, v_2 \in \mathbb{R}$, and $\vec{a_x}, \vec{a_y}$ are orthonormal bases. $v_1 = \vec{v} \cdot \vec{a_x}$, and $v_2 = \vec{v} \cdot \vec{a_y}$. We note then that v_1, v_2 are known as the coordinates, or components of v. Recall that

$$\vec{v} = \begin{pmatrix} \vec{v} \cdot \vec{a_x} \\ \vec{v} \cdot \vec{a_y} \end{pmatrix}_{\vec{a_{xy}}} = \begin{pmatrix} \vec{v_1} \\ \vec{v_2} \end{pmatrix}_{\vec{A}}.$$

To convert this to another basis, we utilize a basis conversion table, and note that

$$\vec{b_x} = \left(\vec{b_x} \cdot \vec{a_x}\right)\vec{a_x} + \left(\vec{b_x} \cdot \vec{a_y}\right)\vec{a_y}.$$

We label the horizontal rows as $\vec{b_x}, \vec{b_y}$ and vertical columns as $\vec{a_x}, \vec{a_y}$. We then get the rotation transformation matrix of

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}.$$

12 October 12, 2016

12.1 Basis Conversion Cont'd

We call the above transformation matrix $\mathbb{R}^{AB}$. Then we note that to re-write a vector $\vec{v}$ in basis A to basis B, we have

$$[\vec{v}]_A = \mathbb{R}^{AB}\,[\vec{v}]_B\,.$$

We note the interesting property that the inverse of R^{AB} is simply its transpose, which we denote as $\mathbb{R}^{BA}$. That is, we have an analogous way to re-write a vector in basis B to basis A:

$$[\vec{v}]_B = \mathbb{R}^{BA}\,[\vec{v}]_A\,.$$

12.2 Reference Frames

A **reference frame** is a vector basis (preferably orthonormal) and an origin (reference point). We can describe vectors in terms of two points P and A. That is, the vector

$$\vec{r}^{P/A}$$

is the vector from point A to point P. We can perform vector addition in this way and note that if we are given another vector from point B to P, we have

$$\vec{r}^{P/A} = \vec{r}^{B/A} + \vec{r}^{P/B}.$$

We call this process **translation** or **displacement**.

For vectors expressed as points in different bases, we recall that we cannot add vectors between these bases. Thus, we need the transformation matrix as follows

$$\left[\vec{r}^{P/A}\right]_A = \left[\vec{r}^{B/A}\right]_A + \mathbb{R}^{AB}\left[\vec{r}^{P/B}\right]_B.$$

In the reverse direction, we have

$$\left[\vec{r}^{P/B}\right]_B = \mathbb{R}^{BA}\left(\left[\vec{r}^{P/A}\right]_A - \left[\vec{r}^{B/A}\right]_A\right).$$

12.3 Transformation Matrices

In computer graphics, it is convenient to express information in matrices. In geometric form for instance, we have

$$\vec{w} = s\vec{v}, \quad s \in \mathbb{R}.$$

This can be expressed in matrix form as

$$[\vec{w}] = S\,[\vec{v}]\,, \quad S \in \mathbb{R}^{2\times 2}.$$

We recall that in this case

$$S = \begin{vmatrix} s & 0 \\ 0 & s \end{vmatrix}$$

If we need to scale down or up by a constant factor, we simply adjust the value of s in the matrix above. Many transformations are possible, including uniform scaling, non-uniform scaling, rotation, shearing, and flipping. The 2D transformation matrices are summarized below:

1. **Scaling** by a factor of s is accomplished by the matrix

$$S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}.$$

2. **Rotation** by an angle of θ is accomplished by the matrix

$$R^{AB} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}.$$

3. **Flipping** by horizontally about the x-axis is accomplished by the matrix

$$F_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

while flipping vertically about the y-axis is accomplished by the matrix

$$F_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

12.4 Composing Transformations

As with ordinary matrix multiplication, when composing transformations, attention must be paid to the order in which the transformations are carried out since order matters. While matrix multiplication gives a linear transform, we need an affine transform in order to include translation. A trick that is used involves the addition of an extra homogeneous coordinate to our vectors and matrices,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & u \\ c & d & v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The result of the matrix multiplication gives us the affine transform of

$$x' = ax + by + u$$

$$y' = cx + dy + v$$

To convert between two references frames A and B, we make use of the homogeneous transformation matrix that arises from combining the rotation matrix with the transformation matrix.

$$T^{AB} = \begin{bmatrix} R^{AB} & \vec{r}^{B/A} \\ 0 & 1 \end{bmatrix}$$

where free vectors and bound (position) vectors are encoded respectively as

$$[\vec{v}] = \begin{bmatrix} v_1 \\ v_2 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} \vec{r}^{P/B} \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ 1 \end{bmatrix}$$

13 October 14, 2016

13.1 Bezier Curves

To represent freeform shapes, we need a system that provides an accurate, complete, and indisputable definition of freeform shapes. We recall that a parametric segment obtained by linear interpolation may be described as

$$\begin{aligned} p(u) &= lerp(p_0, p_1, u) \\ &= (1-u)p_0 + up_1 \end{aligned}$$

Quadratic Bezier curves can be described by de Casteljau's Algorithm,

$$\begin{aligned} p_0^1 &= lerp(p_0, p_1, u) \\ p_1^1 &= lerp(p_1, p_2, u) \\ p(u) &= lerp\left(p_0^1, p_1^1, u\right) \end{aligned}$$

or by Bernstein Polynomials,

$$p(u) = (1-u)^2 p_0 + 2u(1-u)p_1 + u^2 p_2.$$

13.2 Properties of Bezier Curves

The following are important properties of Quadratic Bezier curves.

1. **Endpoint Interpolation**: The endpoints of the curve are always the first and last points specifying the curve. That is, $p(0) = p_0$ and $p(1) = p_2$.

2. **Endpoint Tangents**: The tangent at the endpoints is the line that extends from the endpoints to the middle point. That is, $p'(0) = 2(p_1 = p_0)$ and $p'(1) = 2(p_2 - p_1)$.

3. **Convex Hull**: The curve is always within the convex hull that is formed between all three points (the triangle that results from connecting the three points).

14 October 17, 2016

14.1 Bezier Splines

Bezier splines are just curve segments that are joined together. That is, for two curves defined by the points p_0, p_1, p_2 and q_0, q_1, q_2, a spline would result when $p_2 = q_0$. We note that there are varying levels of parametric and geometric continuity that result.

- C^0 continuity results when the spline is continuous where $p(1) = q(0)$.

- C^1 continuity results when the spline is smooth with the tangent at the intersection being equal in direction and magnitude where $p'(1) = q'(0)$.
- G^1 continuity results when the spline is smooth with the tangent at the intersection being equal in direction where $p'(1) = sq'(0), \quad s \in \mathbb{R}^+$.

14.2 Cubic Bezier Curves

Cubic Bezier curves are defined synonymously, with the Bernstein form being given as

$$p(u) = (1-u)^3 p_0 + 3u(1-u)^2 p_1 + 3u^2(1-u)p_2 + u^3 p_3.$$

Note that the expression for Bezier Curves of the nth degree can be obtained by evaluating Pascal's triangle. More specifically, it is given by the expression

$$p(u) = \sum_{i=0}^{n} b_{i,n}(u) p_i,$$

where

$$b_{i,n} = \binom{n}{i} u^i (1-u)^{n-i}.$$

All of the three previous properties still hold for a Bezier Spline comprised of Cubic Beziers. Additionally, we have the property of curve subdivision, which allows for the identification of the location of a point on the curve given its parameters.

15 October 19, 2016

15.1 Typography

A **typeface** is how the characters are supposed to look, whereas a **font** defines the characteristics of the typeface.

15.2 History of Type

The Greek alphabet existed since around 800 *B.C.* In the first millennium, scribes would reproduce written text. We note that the letterforms have changed throughout time. Johannes Gutenberg invented in Gutenberg Press in 1450, and essentially revolutionized the reproduction of text. It is based on the concept of movable type. We also note that this is where the concept of "cases" arose, since the upper case letters were stored in the upper drawer, whereas the lower case letters were stored in the lower drawer. What was the cause of discrepancy in the quality of type with the introduction of computers?

John Warnock was one of the key players in improving the situation. Steve Jobs was also very influential, as he had brought to the Mac beautiful typography.

Warnock and Jobs improved the situation of digital typography, while Bill Gates unintentionally did the field a disfavor.

15.3 Categorizing Typefaces

There are six main styles of typefaces:

1. **Oldstyle** is characterized by its diagonal stress on letters and the slanted serifs on lowercase letters. They also employ moderate thick and thin transitions in the strokes, and a curved bracketing.

2. **Modern** is characterized by its vertical stress on letters and its lack of bracketing. There are radical thick and thin lines along with serifs on lowercase letters that are thin and horizontal.

3. **Slab serif** is characterized by its vertical stress on letters, along with serifs on lowercase letters that are thick horizontal slabs. There is little to no thick and thin transition or contrast in the strokes.

4. **Sans serif** is characterized by its lack of serifs. There are also no thick and thin transitions in the strokes, as well as an apparent lack of stress.

5. **Script** is characterized by its handwritten, cursive style.

6. **Decorative** is characterized by its fancy and ornamental nature.

15.4 Typeface Nomenclature

The **x-height** is the line to which most lower case letters rise to. The **baseline** is the lower line that defines most lower case letters. The **ascender** is the part of letters that ascend above the x-height. The **descender** is the part of the letters that descend below the baseline.

When specifying the font size, we refer to the point. We note that 1 inch is equal to 6 picas or 72 points. This causes discrepancies, since the resulting text may be projected to any arbitrary size. Type size is therefore defined as the size of the body on which the type is theoretically meant to be cast.

Leading is the distance from one baseline to the next. It was named for the lead that was used to physically provide space between movable type. **Spacing** defines the space taken up by each letterform or space. A **family** of letterforms refers to the entire collection of typefaces that vary in some respect. Ie: bold face, italics, oblique, small caps, text figures, etc. **Weight** refers to the width of the lines of the type. **Kerning** is the grooming of a font so that the spacing between the letters are pleasing.

16 October 21, 2016

16.1 Digital Type

When kerned properly, characters may run into each other. For these instances, a single block was carved for more than one letter. When **ligatures** are applied, the characters are combined and set together.

16.2 Representations in Digital Form

ASCII and Unicode are mappings of numbers to characters. A **glyph** is a typographic term for a shape. It can be defined as a series of bezier curves connected into a bezier spline. We note that a character refers to the letter itself, whereas a glyph refers to the shape of the letter. Thus, fonts provide glyphs for characters. We may have many glyphs to characters or many characters to glyphs mapping. For instance, when dealing with accents or ligatures.

Kerning tables are used to determine how to space pairs of letters when rendering fonts. All of this information including bezier curves and splines, character mappings with glyphs, and kerning tables are stored in files. The TrueType format created by Apple/Microsoft in the late 1980s was the first attempt to store font information using Quadratic Beziers. Adobe Type 1 by Adobe Systems was created in 1984, and OpenType by Microsoft/Adobe in 1996 (which uses Cubic Beziers).

16.3 Rendering Text with Digital Type

Before the advent of representing fonts through Bezier curves, each pixel would be turned on or off to represent text. Today, we have a bunch of points that we use bezier curves on. By approximating with Bezier curves, we can obtain the outline of a particular character. To draw solid glyphs, we can approximate using triangles or raster filling. The size of glyphs is generally measured in 1000 "font units", with the **advance** referring to the length that the glyph takes up and where the next may start to be drawn. Many fonts in 1990s used manual hinting to account for low resolution displays. The basic text rendering algorithm would therefore consist of loading a font (and optionally choosing a variant), obtaining a composite glyph for a character (and optionally checking for ligatures), drawing the Bezier curves, advancing the glyph by the specified amount (and optionally checking the kerning table), and then repeating this procedure for the next character.

16.4 OpenGL Graphics Pipeline

Data travels through the Vertex Specification, the Tessellation Control Shader, the Tessellation Evaluation Shader, the Geometry Shader, Rasterization, the Fragment Shader, and the Frame Buffer.

1. **Vertex Specification** deals with how the vertices are to be drawn. For instance, GL_POINTS, GL_LINES, $GL\_TRIANGLES$, and $GL\_PATCHES$. This is done by sending it vertices, which is processed into attributes.

2. **Vertex Shader** allows one to scale and translate the vertices, and then send these transformed vertices to the next shader.

3. **Tessellation Control Shader** specifies the tessellation level. For instance, $TessellationOuter[0] = 1$ gives continuous line, while $Tessellation[1] = 30$ means to approximate the character with 30 lines. Tessellation is essentially a way to tell OpenGL to not just draw points, lines and triangles. This shader sends patch vertices and attributes to the next shader.

4. **Tessellation Evaluation** calculates the positions and attributes of the generated vertices, and then sends the tessellated vertices to the Geometry Shader. The tessellation evaluation shader specifies the tessellation primitive type. Layout (isolines) in Assignment 3. There is one invocation per vertex, so a unique (u, v) coordinate is for

$$u = gl\_TessCoord.x$$

$$v = gl\_TessCoord.y.$$

The goal is to calculate a new vertex position $gl\_Position = P(u)$. Furthermore, all patch vertices can be accessed by

$$gl\_in[i].gl\_Position$$

where $i \in [0, ..., patchSize]$.

5. **Geometry Shader** is primarily optional for the purposes of this course.

6. **Rasterization** converts the vertices to fragments (which can be thought of as similar to pixels) and sends these fragments.

7. **Fragment Shader** computes the colour.

8. **Frame Buffer** prints out the result.

17 October 24, 2016

17.1 Image Formation

The camera obscura (meaning dark room) was a box with a hole in it that projected an image of what is on the opposing side of the hole. We note that the hole inverts the image. The camera obscura made it possible to draw things while incorporating perspective, which can be described as a result of similar triangles. That is, the inverse would give perspective. We can specify one. two, and three point perspective.

The pinhole camera utilized the principles of the camera obscura for photography, and was used up until WWII. In reality, to get a high quality image, one would need an infinitely small hole. The modern camera came about in the 1960s, which used lenses to gather more light. Positive converging lens are used to collect light and focus it on a single point. The **Lens Equation** is used to determine where the image would be in focus given the distance of the object from the lens:

$$\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f}$$

where S_1 is the distance from the object to the lens, S_2 is the distance from the the lens to the real image, and f is the focal length. A limitation of using lenses is that any object closer than the focal length cannot be focused.

17.2 Focusing

When a cone of light rays from a lens does not come into perfect focus when imaging a point source, and optical spot called the **circle of confusion** results. The circle of confusion is used to determine the depth of field where a part of an image is reasonably sharp. Since real lenses cannot focus all rays perfectly, even at best focus, a point is imaged as a spot rather than as a point. to adjust the focus on objects located at various distances from the lens, the lens are moved relative to the image or film plane.

We note that according to the lens equation, the size of a picture is determined solely by the other two variables. The goal is to get as much light onto the sensor as possible. Thus, in many modern digital cameras, multiple lens are used to control the focal length. We note that an image is therefore controlled by the focal length, aperture, shutter speed, film sensitivity, and lens characteristics. Because one cannot change the size of film or the image sensor, the longer the **focal length** means moving the film back more. Thus, a longer focal length means a smaller field of view.

18 October 26, 2016

18.1 Anatomy of the Digital Camera

Aperture is measured by the f-number, which is a ratio of the lens focal length to diameter of lens aperture. A larger aperture lets in more light, but the circle gets bigger, thus giving a depth of field effect. A larger circle provides a larger circle of confusion. A larger f-number means a smaller aperture. The **shutter speed**, measured in fractions of a second, changes the blurriness over time, whereas the circle of confusion changes the blurriness over space. **Film sensitivity** changes the amount of light needed to cause a chemical reaction on the film. A higher ISO (measurement of film sensitivity), means less amount of light required to register on film. **Lens characteristics** may cause the colours to refract. That is, chromatic aberration causes the colours to split when passed through the lens. We can attempt to fix this by using an achromatic doublet that tries to account for this difference with the crown and flint on the lens.

18.2 The Digital Revolution

In 1969, W. Boyle and G. Smith at AT&T Bell Labs invented charge-coupled devices. Photons are converted to electrical charge and stored in a capacitor array. More photons result in a greater charge. The charges are read from an array to produce an image. This won the 2009 Nobel Prize in Physics. Active Pixel Sensors are cheaper to make than CCDs. They use a CMOS sensor where each pixel has its own active amplifier that allows for a continuous read. A limitation of CMOS is when we can read the information, whereas CCDs wait for the user to read the information. To capture colour, a Bayer Mosaic Pattern Colour Sensor Grid is used, which matches closely to the colour sensitivity of our retinas through its distribution of red, green, and blue.

Remark. Even though the iPhone 6s camera has more megapixels, the Nikon D100 camera has a larger lens and a larger sensor.

19 October 28, 2016

19.1 Ray Tracing

Ray tracing is a method for synthesizing images of virtual 3D scenes. Our goal is therefore to simulate the camera obscura, as opposed to a lens camera. The general ray tracing algorithm involves performing the following steps for each pixel or position on the image:

1. **Ray generation** involves determining where one is looking.
2. **Ray intersection** concerns the calculation of whether one sees anything in the specified direction.
3. **Shading** determines the colour which is seen.

19.2 Ray Generation

To generate rays for a given scene, we arbitrarily establish a virtual image plane between the scene and the camera. We then assign a position vector for each pixel that would appear on this screen from the camera, given by

$$r(t) = e + t(s - e)$$

where $r(t)$ specifies the view ray, $0 \leq t \leq 1$ is the time, e is the position of the camera, and s is the position of the pixel on the screen.

20 October 31, 2016

20.1 Ray Intersection

For each object in our scene, we need to determine whether we can see the object. This involves solving an equation of one variable. Given a ray $r(t) = o + td$, we can determine its intersection with spheres, planes, and triangles.

A **sphere** is given by the parametric equation

$$(p - c) \cdot (p - c) - R^2 = 0$$

where c is center and R is the radius. Alternatively, this can be expressed as

$$\|p - c\| = R.$$

By solving for t, we can substitute this value back into the equation of the ray to determine the intersection. **Ray-sphere intersection** is found from solving

$$(o + td - c) \cdot (o + td - c) - R^2 = 0.$$

A **plane** is given by the parametric equation

$$(p - q) \cdot \hat{n} = 0$$

where q is a point on the plane, and $\hat{n}$ is a unit vector normal to the plane at point q. **Ray-plane intersection** is therefore

$$(o + td - q) \cdot \hat{n}.$$

A **triangle** is represented using barycentric coordinates. Any point in the triangle can be expressed as a linear combination of the three vertices of the triangle. Given points p_0, p_1, and p_2 that form the vertices of a triangle, we can express barycentric coordinates as

$$f(u, v) = (1 - u - v)p_0 + up_1 + vp_2.$$

To determine the coordinates of a point expressed in barycentric coordinates, we simply equate the point to this expression and solve for t, u, and v. To accomplish this, we realize that a triangle necessarily lies on a plane. Given the three vertices (which lie on this plane), we may obtain a normal at any point by calculating the cross product of the vector between this point and the others. For instance, taking the center point q as p_0, the normal $\hat{n}$ is given by

$$(p_2 - p_0) \times (p_1 - p_0).$$

For **ray-triangle intersection**, we first check whether the ray intersects the plane on which the triangle lies. We then equate this point with the expression for a triangle to obtain

$$o + td = (1 - u - v)p_0 + up_1 + vp_2$$

and solve for u, v, and $1-u-v$. If all these values are between 0 and 1 inclusive, then the point lies within the triangle. Otherwise, the point lies outside of the triangle and thus the ray does not intersect the triangle.

21 November 4, 2016

21.1 Materials and Shading

The appearance of a material is determined by the manner in which light is reflect from it, or transmitted through it.

A **shadow ray** is cast from the point of intersection of the view ray with an object in the scene. It extends from this intersection point to the light source. If it intersects any other object before reaching the light source, then this point is in a shadow.

Specular reflection refers to the mirror-like reflection of light from a surface, where a single incoming ray is reflected into a single outgoing ray. When ray tracing ideal specular reflections, we trace another ray as if we were looking from the surface of the material. The reflected ray r, is given by

$$r = d - 2\left(d \cdot \hat{n}\right)\hat{n}$$

where d is the view ray directed at the surface, and n is the surface normal. However, real surfaces are often not perfectly specular. To account for this, we usually combine the reflected ray's light with the surface material colour.

To ray trace transparent materials such as a glass marble, we make use of Snell's Law, which states that

$$\frac{\sin(\theta)}{\sin(\phi)} = \frac{v}{v_t} = \frac{n_t}{n}.$$

By referring to the index of refraction of common materials, we can ray trace refraction.

Diffuse reflection is the reflection of light from a surface such that the incident ray is reflected at multiple angles. The visibility of objects is primarily caused by diffusely scattered light. The brightness of diffuse reflection is dependent only on the direction of the incident light. Thus, the view direction has no effect on the appearance. Using Lambert's cosine law, diffuse reflection is given by

$$c = c_r c_l \max\left(0, \hat{n} \cdot \hat{l}\right).$$

22 November 7, 2016

22.1 Materials and Shading Cont'd

Specular highlights refers to the bright spot of light that appears on shiny objects when they are illuminated. There are two formulas which could be used to account for specular highlights. The first is the **Phong lighting model** given by

$$c = c_l \max\left(0, \hat{e} \cdot \hat{r}\right)^p$$

where e is the view ray in the opposite direction, and $r = -l + 2\left(l \cdot \hat{n}\right)\hat{n}$. The second is the **Blinn-Phong lighting model** given by

$$c = c_l \max\left(0, \hat{h} \cdot \hat{n}\right)^p$$

where $\hat{h} = (e + l)/\left(\|e + l\|\right)$

Remark. We recall that $a \cdot b = \|a\|\|b\| \cos(\theta)$. For unit vectors, the magnitude is 1, so we can substitute $\cos(\theta)$ into the expressions for Phong and Blinn-Phong lighting, where θ is the angle between the two vectors. The shinier the object, the greater the phong exponent p.

Ambient light refers to the light that is reflected off of objects in the scene. It is a form of indirect lighting that allows one to produce soft shadows. It is approximated by a constant light source that represents the average indirect light energy in the scene, and is given by

$$c = c_r c_a$$

With ambient, diffuse, and specular terms considered, the final shading equation becomes

$$c = c_r\left(c_a + c_l \max\left(0, \hat{n} \cdot \hat{l}\right)\right) + c_l c_p \left(\hat{h} \cdot \hat{n}\right)^p$$

where c_r is the diffuse reflectance colour of the material, c_p is the specular colour of the material, p is the Phong exponent representing the shininess of the material, c_l is the light source intensity colour, and c_a is the ambient light intensity colour.

However, there are materials that we still cannot model simply. This includes materials such as milk and skin, which require **subsurface scattering**. Hair, brushed metal, and fire are among the materials for which there is no simple means to calculate shading. **Environment mapping** with a mirrored sphere is a simple way to give reflective materials a realistic rendered appearance.

23 November 14, 2016

23.1 Texture Mapping

Even with a realistic shading model, our motivation for texture mapping comes from the desire to define high frequency detail, surface texture, and colour information.

Texture mapping is the use of an image to store and render spatially varying surface properties. The problem of texture mapping is divided into three sub-problems, including defining a texture mapping function, looking up the image values through sampling, and finally modifying the surface properties.

The mapping function is defined from object space to texture space, and is given by

$$\begin{aligned}\phi &: S \mapsto T \\ &: (x, y, z) \mapsto (u, v)\end{aligned}$$

In defining a mapping function, we want a one-to-one function with **bijectivity**. This allows for optimal control, since every surface point maps to a different texture image position. This ensures that one texture value does not have influence of the properties of multiple surface points. We also aim to minimize **size distortion** since we want the scale of the texture to be approximately constant across the surface. That is, the derivatives of ϕ should not vary greatly. This allows for a level of detail which is consistent over the object, and ensures that it is more intuitive to create and interpret the texture. **Shape distortion** results when a shape drawn on the object does not map to the same shape in the texture. To minimize shape distortion, the derivatives of ϕ should be similar in all direction. **Continuity** is also favoured, since the goal is to have as few seams as possible. Continuity is often a trade-off with the other characteristics. When it is not possible to maintain continuity, seams are place in inconspicuous places in the model.

Some common mapping functions for parametric surfaces include **planar projection** where $u = x$ and $v = y$, or

$$\phi(x, y, z) = (x, y),$$

spherical mapping where

$$\phi(x, y, z) = \begin{bmatrix} \frac{\pi + \tan^{-1}\left(\frac{y}{x}\right)}{2\pi} \\ \frac{\pi - \cos^{-1}\left(\frac{z}{\|x\|}\right)}{\pi} \end{bmatrix},$$

and **cylindrical mapping** where

$$\phi(x, y, z) = \begin{bmatrix} \frac{\pi + \tan^{-1}\left(\frac{y}{x}\right)}{2\pi} \\ \frac{1+z}{2} \end{bmatrix}.$$

For complex shapes, hand-crafted mapping functions are required. To texture a triangle, we use **barycentric mapping**. Given a triangular area in texture space defined by three points t_0, t_1, and t_2, we define the mapping function as

$$(u, v) = (1 - u - v)t_0 + ut_1 + vt_2.$$

24 November 16, 2016

24.1 Texture Sampling

Anisotropic filtering is a method for enhancing the image quality of textures on surfaces of computer graphics that are at oblique viewing angles with respect to the camera where the projection of the texture (not the polygon or other primitive on which it is rendered) appears to be non-orthogonal.

To properly texture map, we need to consider the ratio of texture pixels, or **texels**, to the pixels that appear on the screen. That is, we need to determine whether the texels from the texture map are larger or smaller than the pixels of the rendered image. In the first case, we have **texture magnification**, where one texel is mapped to many pixels. This becomes a problem of interpolation, as we need to reconstruct the image. **Texture minification** occurs when many texels are mapped to one pixel. Since we are now fitting many texels to one pixel location, problems of aliasing occur. Thus, we solve this by anti-aliasing textures by blurring to reduce noise in the image.

Texture filtering is a method used to determine the texture colour for a texture mapped pixel using the colours of nearby texels. Alongside anisotropic filtering, there are a variety of filtering methods. Among them is **mipmapping**, which is based on constructing layers, each half the size of the previous, to form an image pyramid. The appropriate pyramid level is chosen depending on the footprint of the texel on the pixel. In the process of minification, we apply mipmapping to reduce the number of texels read by prefiltering the texture and storing it in smaller sizes. As the texture surface moves farther away, we then make use of the appropriate level to avoid the expensive operation of reading all texels and combining their values. Thus, depending on the pixel, we must choose the right strategy for upsampling/magnification, or for downsampling/minification.

24.2 Surface Properties

There are many elements which can be interpreted as texture maps. For instance, colour or diffuse maps specify the colour, specular maps specify the highlights, normal maps specify the normal vector at each location, bump maps specify the bumps in the material, and light maps specify the lighting at a particular location. Thus, we can see than texture mapping can be used to modify any surface property that appears in the lighting equation:

$$c = c_r \left(c_a + c_l \max \left(0, \hat{n} \cdot \hat{l} \right) \right) + c_l c_p \left(\hat{h} \cdot \hat{n} \right)^p .$$

25 November 18, 2016

25.1 3D Vectors Review

We reviewed basic vectors and vector operations in 3D.

26 November 21, 2016

26.1 3D Transformation Matrices

The 3D transformations are simply an extension of their 2D counterparts. Some important 3D transformation matrices are summarized below:

1. **Scaling** by a factor of s is accomplished by the matrix

$$scale(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}.$$

2. **Rotation** is now more complicated, since there are different axes of rotation. Rotating along each of the x, y, and z axes is accomplished by each of the respective matrices

$$rotate - x(phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix},$$

$$rotate - y(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix},$$

$$rotate - z(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

In 3D, we can also rotate about any arbitrary vector $\vec{a}$. We first need to form an orthonormal basis uvw where $w = \vec{a}$. A rotation along this arbitrary vector can be thought of as a rotation from this basis to the canoncial xyz basis, a rotation about the z-axis, then a rotation from the canonical basis back to the uvw basis. A rotation about the w-axis by an angle of ϕ is accomplished by

$$\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}.$$

3. **Translation** in 3D is accomplished by adding a fourth homogeneous coordinate, where
$$\begin{bmatrix}1 & 0 & 0 & x_t\\ 0 & 1 & 0 & y_t\\ 0 & 0 & 1 & z_t\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}=\begin{bmatrix}x+x_t\\ y+y_t\\ z+z_t\\ 1\end{bmatrix}.$$

We note that 3D transformation matrices can be composed in the same manner as 2D transformation matrices. We can also easily determine the inverse transformations. The inverse of $scale(s_x, s_y, s_z)$ is $scale(1/s_x, 1/s_y, 1/s_z)$. The inverse of a rotation is its transpose, and the inverse of a translation is a translation in the opposite direction.

27 November 23, 2016

27.1 Coordinate Transformations

These transformation matrices which are used to move points around can also be thought of as changing the coordinate system in which a point is represented. We have the following matrices to convert between coordinate systems in 2D. The 3D case is performed analogously.
$$\begin{bmatrix}x_p\\ y_p\\ 1\end{bmatrix}=\begin{bmatrix}1 & 0 & x_e\\ 0 & 1 & y_e\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_u & x_v & 0\\ y_u & y_v & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}u_p\\ v_p\\ 1\end{bmatrix},$$
$$\begin{bmatrix}u_p\\ v_p\\ 1\end{bmatrix}=\begin{bmatrix}x_u & y_u & 0\\ x_v & y_v & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -x_e\\ 0 & 1 & -y_e\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_p\\ y_p\\ 1\end{bmatrix}.$$

28 November 25, 2016

28.1 Scene Graphs

When objects in a scene are transformed relative to one another, it is most appropriate to represent this in a scene graph. Suppose we have transformation matrices M_1 for the first object in relation to the canonical basis, M_2 for the second object in relation to the position of the first object, and M_3 for the third object in relation to the position of the second object. The appropriate matrix to apply to an object would therefore be the product of all the matrices in the chain from the object to the root of the data structure. That is, the first object is transformed using M_1, the second using M_1M_2, and the third $M_1M_2M_3$.

28.2 View Position

To change our viewpoint in 3D to look in any direction, we make use of the following convention to specify the view position and orientation. Given a camera position e, a view direction g, and an up vector t, we can construct a right handed basis with w pointing in the opposite direction of g, and v in the same plane as g and t. The camera position dictates the location that one views the scene. The view direction specifies the direction that the viewer is looking. The up vector t specifies the vector that points directly upwards "to the sky". With these three vectors, we can form a uvw basis, where

$$\begin{aligned} w &= -\frac{g}{\|g\|}, \\ u &= \frac{t \times w}{\|t \times w\|}, \\ v &= w \times u. \end{aligned}$$

In the event that the points we wish to transform are stored with the canonical origin along the xyz axes, we would need to convert between the two bases.

29 November 28, 2016

29.1 Perspective Transform

To capture the effects of perspective, we need a way to be able to draw line segments smaller if they are farther from the viewer than similar line segments that are closer. The size of an object on the screen in proportional to $1/z$. We note that objects at twice the distance from the camera appear at half of the height. That is,

$$\begin{aligned} x_p &= \frac{x}{-z}, \\ y_p &= \frac{y}{-z}. \end{aligned}$$

This allows us to relate to the focal distance and the field of view by letting

$$\begin{aligned} x_p &= \frac{dx}{-z}, \\ y_p &= \frac{dy}{-z}, \\ \tan\left(\frac{\theta}{2}\right) &= \frac{1}{d}, \end{aligned}$$

where d is the distance from the camera to the virtual image plane with a width from -1 to 1. Thus, perspective projection is given by

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} dx \\ dy \\ 1 \\ -z \end{bmatrix} \to \begin{bmatrix} \frac{dx}{-z} \\ \frac{dy}{-z} \\ \frac{1}{-z} \end{bmatrix}.$$

30 November 30, 2016

30.1 Hidden Surface Removal

The **Painter's Algorithm** is used to sort object primitives in order from back to front. Thus, we render the primitives in sorted order, over the existing scene. We note that there are situations when this does not work. For instance, when something from behind overlaps another object in the front. Another case would be when an object in the front intersects something behind it. Projective rendering may create occlusion ordering problems.

The **Z-Buffer Algorithm** solves this issue by associating with each fragment a colour and a depth value. We can modify the projection matrix

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

to implement z-buffering. In the original, we have

$$z_p = -\frac{1}{z}.$$

By modifying the projection matrix for z-buffering, we obtain a final perspective projection matrix of

$$\begin{bmatrix} \frac{d}{a} & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & -\frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

where a is the aspect ratio of width to height. Thus,

$$z_p = \frac{2fn}{z(n-f)} - \frac{n+f}{n-f}.$$

With the z-buffer algorithm, we can compute a depth value for each fragment using the projection matrix. If the depth of the fragment is less (greater) than the existing depth stored at (x_p, y_p), then we write the fragment colour to the frame buffer. Otherwise, the fragment is discarded.

A **view frustum** is defined by the camera position, field of view, aspect ratio, near plane, and far plane. It forms a truncated 3-dimensional pyramid shape that encloses everything we can see in our 3D scene. The view frustrum defined by the projection matrix, allows us to encapsulate everything visible in our 3D scene. We note that while perspective projection with the z-buffer algorithm is simple and fast, it does not always work. For instance, when transparency has to be taken into account. In the case that transparency is required, we make use of the painter's algorithm.

Remark. A pixel is a unit that is shown on the screen. A fragment however, may not show up on the screen. For instance, a fragment is created internally and processed, and may ultimately end up hidden.