2D rotation: P'= $\begin{pmatrix} cos & -sin \\ sin & cos \end{pmatrix}\begin{pmatrix} Px \\ Py \end{pmatrix}$ rotation is about the origin

2D scaling: P'= $\begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix}\begin{pmatrix} Px \\ Py \end{pmatrix}$ uniform scaling if Sx= Sy; 2D Translation: $\begin{pmatrix} Px + Vx \\ Py + Vy \end{pmatrix}$

Homogeneous Representation of 2D Transformations:

$\mathbf{p}' = \mathbf{S}\mathbf{p}$

$$\begin{pmatrix} p'_x \\ p'_y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x p_x \\ s_y p_y \\ 1 \end{pmatrix}$$

$\mathbf{p}' = \mathbf{S}\mathbf{p}$

$$\begin{pmatrix} p'_x \\ p'_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}$$

$\mathbf{p}' = \mathbf{T}\mathbf{p}$

$$\begin{pmatrix} p'_x \\ p'_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + v_x \\ p_y + v_y \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} p_x \cos\theta - p_y \sin\theta \\ p_x \sin\theta + p_y \cos\theta \\ 1 \end{pmatrix}$$

Now a translation can be represented as a matrix as well

Note that translation has no effect on vectors.

:3D Transformations

Rotation about x-axis:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about y-axis:

$$\mathbf{R}_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about z-axis:

$$\mathbf{R}_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Fixed point of a 3D rotation is a straight line

Scaling

$$\mathbf{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

inverse of T?

$$\mathbf{T}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -v_x \\ 0 & 1 & 0 & -v_y \\ 0 & 0 & 1 & -v_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse Transformations

$$\mathbf{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

inverse of S?

$$\mathbf{S}^{-1} = \begin{pmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Unit Quaternions:

W^(2)+V·V=1

$$\mathbf{q} = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\mathbf{r})$$

$$\mathbf{R}_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

inverse of $\mathbf{R}_z$?

$$\mathbf{R}_z^{-1} = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

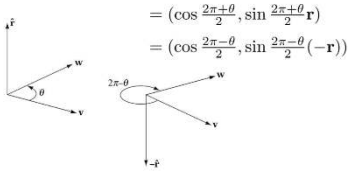
$$= \mathbf{R}_z^T$$

Inverse of a rotation equals its transpose

r is a unit vector represent the axis of rotation. ⊖ is the angle of rotation.

e.g. A rotation about the z-axis by 90 degrees (leftmost)

$$-\mathbf{q} = (-\cos\tfrac{\theta}{2}, -\sin\tfrac{\theta}{2}\mathbf{r})$$
$$= (\cos\tfrac{2\pi+\theta}{2}, \sin\tfrac{2\pi+\theta}{2}\mathbf{r})$$
$$= (\cos\tfrac{2\pi-\theta}{2}, \sin\tfrac{2\pi-\theta}{2}(-\mathbf{r}))$$



For addition and scalar multiplication, a quaternion behaves like a 4-vector.

$$w = \cos\left(\frac{\pi}{4}\right) = \frac{\sqrt{2}}{2}$$
$$x = 0 \cdot \sin\left(\frac{\pi}{4}\right) = 0$$
$$y = 0 \cdot \sin\left(\frac{\pi}{4}\right) = 0$$
$$z = 1 \cdot \sin\left(\frac{\pi}{4}\right) = \frac{\sqrt{2}}{2}$$

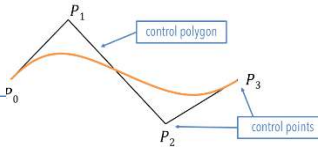
Bézier Curves

线性： P(t)=(1-t)P0+tP1, $t \in [0,1]$； 二次： P(t)=(1-t)^(2)P0+2t(1-t)P1+t^(2)P2; 三次： P(t)=(1-t)^(3)P0+3t(1-t)^(2)P1+3t^(2)(1-t)P2+t^(3)P3; 四次: P(t)=(1-t)^(4)P0+4t(1-t)^(3)P1+6(1-t)^(2)t^(2)P2+4t^(3) (1-t)P3+t^(4)P4

$$P(t) = B_{0,n}(t)P_0 + B_{1,n}(t)P_1 + \cdots + B_{n,n}(t)P_n, t \in [0,1]$$

where

$$B_{i,n}(t) = \frac{n!}{(n-i)!\,i!}(1-t)^{n-i}t^i, \quad i = 0,1,\ldots,n$$



Bin(t): blending functions or basis functions

Properties of Bézier Curves:

1. Any polynomial curve can be put in the Bézier form. 2. A Béziercurve of degree interpolates the two endpoints P0 and Pn. Prove: P(0)=P0, P(1)=Pn; 3. A Bézier curve P(t) of degree n interpolates the two end sides of the control polygon.(left)

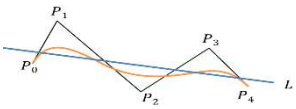
$$P'(t) = \frac{dP(t)}{dt} = n\sum_{i=0}^{n-1} B_{i,n-1}(t)(P_{i+1} - P_i)$$

$$P'(t)|_{t=0} = n(P_1 - P_0)$$
$$P'(t)|_{t=1} = n(P_n - P_{n-1})$$

$$\sum_{i=0}^{n} B_{i,n}(t) \equiv 1,$$

$$B_{i,n}(t) \geq 0, \qquad t \in [0,1]$$

4. Convex hull property The curve segment , lies entirely inside the convex hull of all control points (right)



5. Variation diminishing property: The number of intersections between an arbitrary straight line and the Bézier curve P(t), is NOT greater than the number of intersections between the line L and the control polygon of P(t). 6.Invariant form under affine transformations: M: X'=AX+b For any affine transformation , there is

$$\mathbf{M}\big(P(t)\big) = \sum_{i=0}^{n} B_{i,n}(t)\mathbf{M}(P_i).$$

Proof:

$$\begin{aligned} \mathbf{M}\big(P(t)\big) &= AP(t) + b \\ &= \textstyle\sum_{i=0}^{n} B_{i,n}(t)AP_i + \sum_{i=0}^{n} B_{i,n}(t)b \\ &= \textstyle\sum_{i=0}^{n} B_{i,n}(t)(AP_i + b) \\ &= \textstyle\sum_{i=0}^{n} B_{i,n}(t)M(P_i) \end{aligned}$$

Thanks to this property, when a Bézier curve is transformed affinely, we can a) transform many sampled points on the curve directly, or b) transform the control points only and use the transformed control point

yield the same transformed curve, but the latter can b evaluated on the curve.

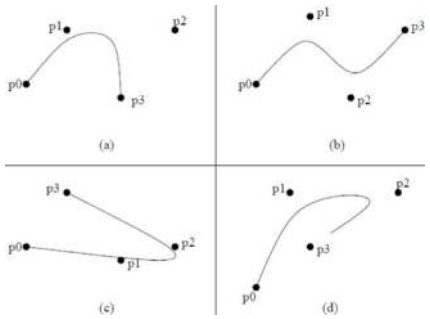
Suppose that a quadratic Bézier curve $P(t)$ is given by $P(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2, t \in [0;1]$; where $P_0 = (0,0)^T$, $P_1 = (4,0)^T$ and $P_2 = (4,4)^T$. Express the curve segment of this curve $P(t)$ over the interval $t \in [0,0.5]$ as a Bézier curve.

Solution: By the de Casteljau algorithm, we have

$$\begin{aligned} P_0^{(0)} &= P_0 = (0,0)^T \\ P_1^{(0)} &= P_1 = (4,0)^T \\ P_2^{(0)} &= P_2 = (4,4)^T \\ P_0^{(1)} &= 0.5P_0^{(0)} + 0.5P_1^{(0)} = (2,0)^T \\ P_1^{(1)} &= 0.5P_1^{(0)} + 0.5P_2^{(0)} = (4,2)^T \\ P_0^{(2)} &= 0.5P_0^{(1)} + 0.5P_1^{(1)} = (3,1)^T \end{aligned}$$

The points $P_0^{(0)}, P_0^{(1)}, P_0^{(2)}$ form the Bézier control polygon of the curve segment of $P(t)$ over the interval $t \in [0,0.5]$, and the curve segement can therefore be expressed as the following Bézier curve

$$\tilde{P}(t) = (1-t)^2P_0^{(0)} + 2t(1-t)P_0^{(1)} + t^2P_0^{(2)}, \ t \in [0,1].$$



A cubic Bézier curve has the end-tangent interpolation property. In Fig (a), the tangent line at P3 is not the line P2P3, so (a) must not be cubic Bézier curves. A cubic Bézier curve must be always in the convex hull of the control points. In Fig (c), a segment of curve is outside line P1P2. A cubic Bézier curve has the endpoint interpolation property. In Fig (d), the end point is not P3, so (d) must not be cubic Bézier curves.

Compute the control points of the cubic Bézier curve representing a segment of the cubic curve $y = 2x^3$, $x \in [-3,3]$.

Solution: The parametric equation of the cubic curve is given by

$$Q(x) = (x, 2x^3), \quad x \in [-3,3].$$

By reparametrization with $t = (x+3)/6$, we obtain the same curve

$$P(t) = (6t-3, 2(6t-3)^3), \quad t \in [0,1].$$

Let P_0, P_1, P_2, P_3 be the 4 control points of $P(t)$.
Due to the end-point interpolating property, we have

$$P_0 = P(0) = (-3,-54)^T \quad \text{and} \quad P_3 = P(1) = (3,54)^T$$

By the end-tangent interpolating property, we have

$$P'(0) = \frac{dP(t)}{dt}\Big|_{t=0} = 3(P_1 - P_0),$$
$$P'(1) = \frac{dP(t)}{dt}\Big|_{t=1} = 3(P_3 - P_2).$$

Now, $P'(t) = (6, 36(6t-3)^2)$ and hence

$$P'(0) = (6, 36\times 9)^T = 3(P_1 - P_0) \Rightarrow P_1 = (-1,54)^T,$$
$$P'(1) = (6, 36\times 9)^T = 3(P_3 - P_2) \Rightarrow P_2 = (1,-54)^T.$$

Let $p'(u) = (x'_p(u), y'_p(u))$ be the first derivative of $p(u)$. Derive the normal of $S(u,\theta)$. (Do not normalize the normal vector in your answer.)

Solution:

$$\begin{aligned} \frac{\partial S(u,\theta)}{\partial u} &= (x'_p(u)\cos\theta,\ y'_p(u),\ -x'_p(u)\sin\theta) \\ \frac{\partial S(u,\theta)}{\partial \theta} &= (-x_p(u)\sin\theta,\ 0,\ -x_p(u)\cos\theta) \end{aligned}$$

The normal of $S(u,\theta)$ is given by

$$\begin{aligned} &\frac{\partial S(u,\theta)}{\partial u} \times \frac{\partial S(u,\theta)}{\partial \theta} \\ &= (-x_p(u)y'_p(u)\cos\theta,\ x'_p(u)x_p(u)\cos^2\theta + x'_p(u)x_p(u)\sin^2\theta,\ x_p(u)y'_p(u)\sin\theta) \\ &= (-x_p(u)y'_p(u)\cos\theta,\ x'_p(u)x_p(u)\ x_p(u)y'_p(u)\sin\theta) \end{aligned}$$

The sequence of OpenGL function calls is: GLfloat M[16] = {1,0,0,0,-0.8,1,0,0,0,0,1,0,0,0,0,1}; glMultMatrix(M); glTranslatef(-0.2,-1.0,0.0); glScalef(0.4,2.0,1.0);

Show that any sequence of rotations and translations can be replaced by a single rotation about the origin followed by a translation. **Solution** We can show by simply multiplying 4×4 matrices that the concatenation of two rotations yields a rotation and then the concatenation of two translations yields a translation. By looking at the product of a rotation and a translation, we find left three columns of RT are the left three columns of R and the right column of RT is the right column of the translation matrix. If we now consider RTR′ where R′ is a 4×4 rotation matrix, the left three columns are exactly the same as the left three columns of RR′ and the right column still has 1 as its bottom element. Thus, the form is the same as RT with an altered rotation (which is the concatenation of the two rotations) and an altered translation. Inductively, we can see that any further concatenations with rotations and translations do no alter this form.

Given an affine transformation $X' = MX + B$, where M is a 2×2 matrix and B is a $2D$ vector, find the equation of the image E of the circle $x^2 + y^2 - 1 = 0$ under this transformation. Show that E is an ellipse.

Solution

We denote the unit circle as $X^TX = 1$, where $X = (x,y)^T$, and define an arbitrary 2D affine transformation $X' = MX + B$, where M is a nonsingular 2×2 matrix, and B is a translation vector of 2×1. So we have $X = M^{-1}(X' - B)$. Substitute it into $X^TX = 1$, and we have

$$[M^{-1}(X' - B)]^T[M^{-1}(X' - B)] = 1$$
$$\Leftrightarrow (X' - B)^T(M^{-1})^TM^{-1}(X' - B) = 1.$$

A rational Bézier curve is represented by

$$P(t) = \frac{\sum_{i=0}^{n} w_i B_{i,n}(t) P_i}{\sum_{i=0}^{n} w_i B_{i,n}(t)}, \qquad t \in [0,1].$$

- w_i can be thought of weights

Sweep Surfaces

Let the cross-section curve be $C(u)$ and the axis curve be $A(v)$.
Let $F(v)$ be the matrix representing a rotation frame attached at $A(v)$. Then the sweep surface is defined by

$$S(u,v) = A(v) + F(v)C(u), \qquad v \in [v_0, v_1].$$

To use $s(v)$ to change the size of the cross-section curve while sweeping it, we have

$$S(u,v) = A(v) + s(v)F(v)C(u), \qquad v \in [v_0, v_1].$$

Now we show that $(M^{-1})^T M^{-1}$ is a symmetric positive definite matrix. This is because $((M^{-1})^T M^{-1})^T = (M^{-1})^T M^{-1}$ and for any nonzero vector Y,

$$Y^T (M^{-1})^T M^{-1} Y = (M^{-1}Y)^T M^{-1} Y = \|M^{-1}Y\|^2 > 0,$$

since M is nonsingular and $M^{-1}Y \neq \mathbf{0}$.
And one definition of ellipse is

$$\{X | (X - X_c)^T A (X - X_c) = 1\},$$

with $X, X_c \in R^n$, X_c the center of ellipse, and $A_{n\times n}$ a symmetric positive definite matrix. Therefore X' is on an ellipse and B is the center.

In OpenGL, antialiasing can be controlled by the second parameter (GL TEXTURE MIN FILTER/GL TEXTURE MAX FILTER) of the texture function glTexParameteri(). Please explain the different effects of GL NEAREST and GL LINEAR? Which one is more suitable for anti-aliasing?
(b)GL NEAREST means to return the value of texture element that is nearest (in Manhattan distance) to the center of the pixel being textured. GL LINEAR will return the weighted average of the four texture elements that are closest to the center of the pixel being textured. GL LINEAR is more suitable for anti-aliasing.
Mipmaps (Image Pyramids) An efficient method for antialiasing of textures. • base of pyramid is the original image (level 0) • level 1 is the image down-sampled by a factor of 2 • level 2 is down-sampled by a factor of 4, and so on • requires that original dimensions be a power of 2 • total size = 4/3 the size of the original image. Mipmaps While texturing, a screen pixel is mapped to the texture space and see how much area it covers. If it is about the same size as a texel, the level 0texture will be used. The larger the area it covers, the higher the texture level in the mipmapis used. Antialiasing with Mipmaps Mipmaps let us efficiently average large regions • each texelin upper levels covers many base texels - at level kthey are the average of 2kx2ktexels • can quickly assemble appropriate texelsfor averaging. Fortunately, OpenGL can take care of most details • gluBuild2DMipmaps()—automatically generate pyramid from base image • control behavior with glTexParameter() • OpenGL handles all filtering details during rasterization

Consider the intersection of a ray with a triangle. The three vertices of the triangle are $A(2,0,2)$, $B(0,3,-2)$, $C(-2,3,2)$. We shoot a ray from the origin in the direction of $(1,1,1)$. Does the ray intersect the triangle? If yes, compute the closest intersection point between them.

Solution: Let P be the plane containing the triangle ABC. Then P is given by

$$N \cdot (X - X_0) = 0,$$

where X_0 is a point on P (we take A as X_0) and N is the normal of P given by

$$\begin{aligned} N &= AB \times AC \\ &= (-2,3,-4)^T \times (-4,3,0)^T \\ &= (12,16,6)^T \end{aligned}$$

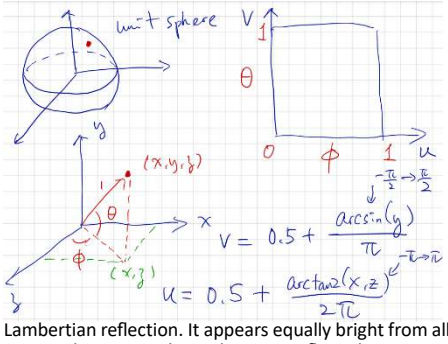
The parametric representation of the ray $R(t)$ is given by

$$R(t) = S + Dt, \quad t \in [0,\infty),$$

where S is the starting point (i.e., the origin), and $D = (1,1,1)$.
Substitute $R(t)$ to the plane equation, and we have

$$t = \frac{N \cdot A}{N \cdot D} = 18/17.$$

Since $t > 0$, we have the intersection point $R(18/17) = (\frac{18}{17}, \frac{18}{17}, \frac{18}{17})$.



(a) Describe the three basic components of Phong illumination equation; (b) Describe how the Flat shading method and Gouraud shading method work; (c) Describe the pros and cons of the Phong shading method.
Solution:
(a) The three basic components of Phong illumination are: diffuse reflection, specular reflection and ambient reflection. Diffuse reflection is also called Lambertian reflection. It appears equally bright from all viewing directions and its intensity is view-direction independent. Its reflected intensity depends only on direction of light source. Specular reflection is the highlight observed on a shiny, glossy or mirror-like surface, such as the surface of glass or plated metal. Its appearance changes as the viewpoint moves. The intensity of specular reflection at a particular point of the surface depends on the viewing direction. Ambient reflection is often modeled by a constant term in a simple reflection equation. It is purely fictitious.
(b) Flat shading method: A single intensity is calculated for each surface polygon. All the pixels in that surface are filled with the same color. Gouraud shading method: Use Phong's illumination equation to compute color intensities at all vertices; Color intensities of pixels along each edge are linearly interpolated from those at vertices; Color intensities of pixels within a span on a scanline are linearly interpolated from the color intensities at the two endpoints.
(c)Pros of the Phong shading method: Phong shading method can be used to provide better highlight rendering result than Gourand shading. Cons of the Phong shading method: This method has to calculate normals for each inside positions, which require more time.

(a) Explain how to detect whether or not a planar boundary face of an object is front-facing. (b) Describe the Z-buffer algorithm briefly, and explain how it is used for hidden surface removal.
Solution: (a) Assume the direction of the view is los, the normal vector of the boundary face is pv. Then the direction of the face can be decided by the value: a = dot(los,pv). If a < 0, which means that the angle between los and pv is among (90◦,270circ), so that the face is front-facing in current view.
(b) The Z-buffer algorithm is used for visual surface determination. First of all, we allocate Z-buffer. Then, set the initial value of it to be at infinity. After that, loop over all of the objects and rasterize the current object. Then, check the z value of each covered pixel(x,y). If it is less than the Z-buffer, set the Z-buffer to z value of the pixel and write pixel. In the rendering process, the depth/Z value of each pixel is checked against an existing depth value. If the current pixel is behind the pixel in the Z-buffer, the pixel is rejected, otherwise it is shaded and its depth value replaces the one in the Z-buffer. Z-buffer supports dynamic scenes easily, and is currently implemented efficiently in graphics hardware.

Getting Rid of Jaggies: The key idea is to use area-weighted sampling
• instead of simply filling in a pixel or not • compute how much of the pixel is covered by the object • and fill in with an appropriately scaled color. When we introduce area-weighted sampling • we transition from solid color jagged objects • to more smoothly colored objects with multiple tones • when viewed from a proper distance is hopefully smoother
Another Kind of Aliasing: Objects may be smaller than pixels
our most common solution is super-sampling • suppose we want to produce a 256x256 image first, generate a 1024x1024 image then down sample to 256x256 by 4x4 averaging • each output pixel is computed from 16 subpixel samples • this reduces efficiency too (by a factor of 16) • but at least we can control it, by the super sampling ratio
Aliasing in Time: • temporal super sampling and average • in other words, motion blur

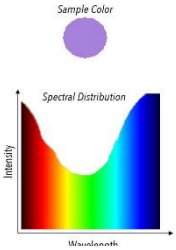
Drawing with Shadow Buffers and OpenGL
Compute and read back the shadow buffer for the light source as well as the projection matrices • UseglReadPixels(…)to read back the depth buffer for the light • UseglGetDoublev(GL_MODELVIEW_MATRIX,…), glGetDoublev(GL_PROJECTION_MATRIX,…), glGetIntegerv(GL_VIEWPORT,…) Render the scene from the viewpoint with lighting and read back both color and depth buffers as well as the projection matrices • we only want to deal with visible surfaces Unprojectevery pixel in the depth buffer to obtain world coordinates Project the world coordinates into the shadow buffer to obtain a z value Compare the current z value with the existing one in the shadow buffer • if the current z value is greater, set pixel in the color buffer= shadow color • otherwise, no change • you will needgluUnProject(), gluProject() Finally, write back the modified color buffer • Use glDrawPixels(…)to write back

From Hard to Soft Shadows Can try to build soft shadows by combining hard shadows • suppose we sample several points on an area light source • treat each one as a point light & compute shadows • now imagine averaging all these shadows together • we get an approximation of the correct soft shadow - objects in umbra are in shadow with respect to all samples - objects in penumbra are only in some of the shadows
Or we can rely on more sophisticated algorithms • For example, radiosity naturally produces soft shadows

Spectral Distribution of Light

"Light" is a mixture of many wavelengths
- each with some intensity
- represented by continuous function $P(\lambda)$ =intensity at wavelength λ
- each as some intensity
- spectral distribution: intensity as a function of wavelength over the entire spectrum

We perceive these distributions as colors
- largely an artifact of our visual system



The Visual System How do we see? • Light travels through cornea, pupil, lens, retina, optical nerves, then brain • Imaging sensors on the retina: rods & cones

Rods and Cones Cones • active at normal light levels • color vision involves cone only Rods • sensitive at low light levels • not sensitive to color • responsible for our dark-adapted vision • low influence on color perception There is an uneven distribution of cones and rods in the retina

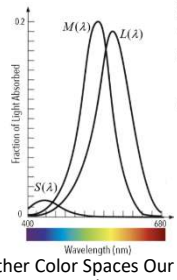
Cone Response

Three kinds of cones: S, L, and M
- S cones respond to blue
- M cones respond to green
- L cones respond to red
- much less sensitive to blue

Response levels to illumination are

$$s = \int S(\lambda) P(\lambda) d\lambda$$
$$m = \int M(\lambda) P(\lambda) d\lambda$$
$$l = \int L(\lambda) P(\lambda) d\lambda$$

This implies that we humans perceive light as a 3D space only because we have 3 cone types

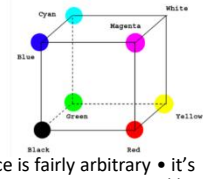


The RGB Color Space

We've represented colors as combinations of three primaries
- established 3 special colors (red, green, blue)
- compose all colors by weighted combinations of these

$$C = rR + gG + bB, \text{ where } (r,g,b) \in [0,1]^3$$

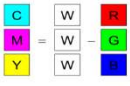
- and why did we pick red, green, and blue?
  - essentially because of the structure of our visual system
  - roughly correspond to peaks of cone response curves



Looking Towards Other Color Spaces Our choice of RGB color space is fairly arbitrary • it's loosely based on our perceptual system We could in principle select any 3 primaries we like • and continue to represent colors as weighted combinations We can also construct other 3-D color spaces • where the dimensions are no longer primary colors • but have some other physical meaning As we'll see, RGB color space is not always the best choice • different color spaces lend themselves to different tasks
Other Primary Colors Red, Green, Blue • liquid crystal, CRT displays Red, Yellow, Blue • paint Cyan, Magenta, Yellow • color printing Orange, Green, Violet • color photography

The CMY Color Space

The other most common set of primaries besides RGB
- cyan, magenta, and yellow — complements of red, green, blue



These are the so-called subtractive primaries
- RGB are additive primaries — start with black, add up to white
  - appropriate when dealing with emitted light
- CMY — start with white and subtract colors from white
  - appropriate when dealing with inks/pigments
  - each ink absorbs some part of the spectrum (subtracts light)

The HSV Color Space

Dimensions no longer primaries
- hue — selects a base color
- saturation — purity of color (decrease = adding white)
- value — brightness (decrease = adding black)

Designed for color specification
- more user-friendly than RGB

Can view this space as a hexcone
- RGB & CMY are cubes