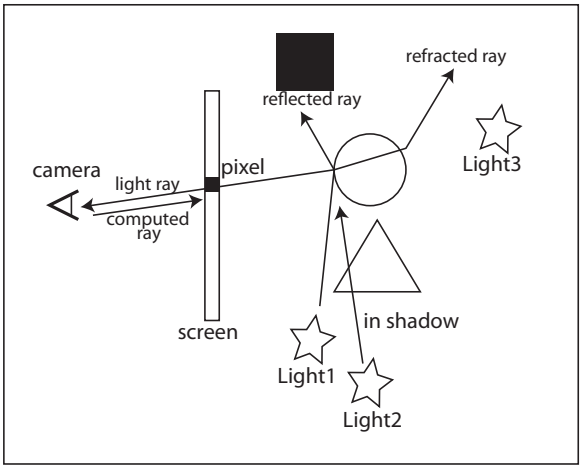
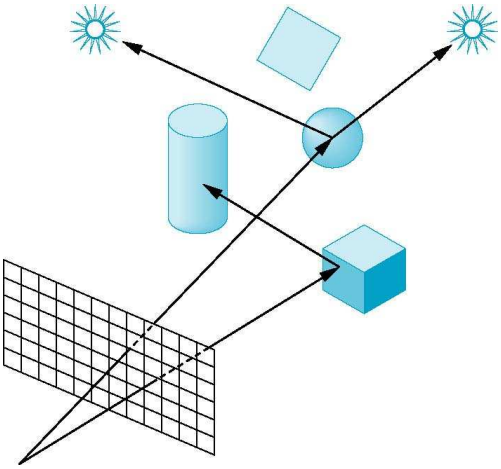
Ray Tracing Notes
CS445 Computer Graphics, Fall 2010

1 The Ray Trace Algorithm





```
For each pixel in image {
    compute ray
    color = Trace(ray)
}

color Trace(ray) {
    For each object
        find intersection (if any)
        check if intersection is closest
    If no intersection exists
        color = background color
    else for the closest intersection
        for each light  // local color
            if (! inShadow(shadowRay)) color += color_local
        if reflective
            color += k_r Trace(reflected ray)
        if refractive
            color += k_t Trace(transmitted ray)
}

boolean inShadow(shadowRay) {
    for each object
        if object intersects shadowRay return true
    return false
}
```

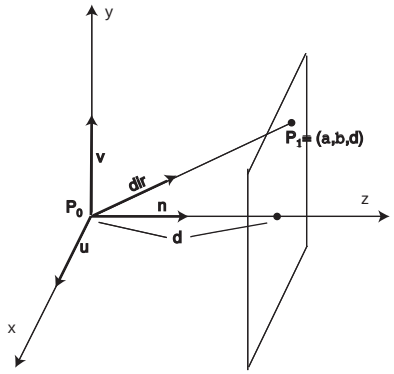
1.1 Complexity

w = width of image
h = height of image
n = number of objects
l = number of lights
d = levels of recursion for reflection/refraction

Assuming no recursion or shadows $O(w * h * (n + l * n))$. How does this change if shadows and recursion are added? What about anti-aliasing?

2 Computing the Ray

We define a ray by 1) a starting point P_0 (at the camera) and 2) a direction defined by the line connecting the camera position and the pixel on the screen, P_1.



$$\begin{aligned} dir &= \text{unit vector in direction from } P_0 \text{ to } P_1 \\ &= \frac{P_1 - P_0}{||P_1 - P_0||}. \end{aligned}$$

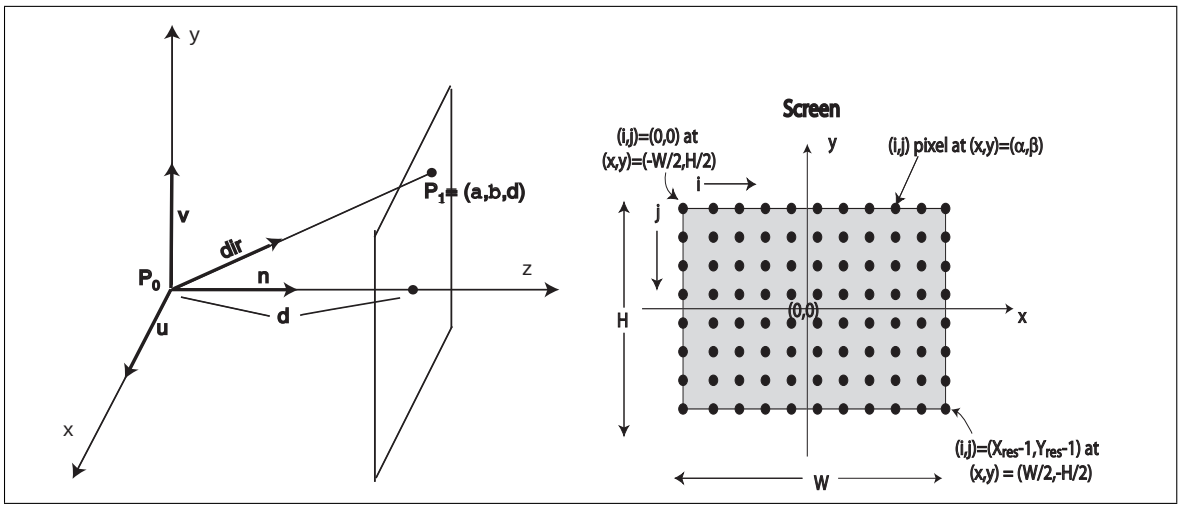
The ray can then be expressed *parametrically* as

$$P = P_0 + t\ dir$$

where $t = 0$ corresponds to P_0. Of course, the goal is to find the closest object that intersects this ray. This involves finding the t value of the all of the intersections and then taking the smallest positive t.

2.1 Computing the Pixel Location in World Space (P_1)

If we are given the row and column (i,j) of the pixel on the screen, what is its location in world space? We first need to identify the location and orientation of the camera and the location, size, and resolution of the screen (pixels).



Given the following:

$$
\begin{aligned}
P_0 &= \text{location of camera} = (0,0,0). \text{ Looks down } \hat{n}. \\
\hat{n} &= \text{unit vector normal to view plane, in picture } = (0,0,1) \\
\hat{v} &= \text{unit vector in up direction, in picture } = (0,1,0) \\
\hat{u} &= \text{unit vector in right direction (left-handed system), in picture } = (1,0,0) \\
d &= \text{distance of camera from view screen in World Coordinates (WC).} \\
H &= \text{height of screen in WC.} \\
W &= \text{width of screen in WC} \\
X_{res} &= \text{number of pixels per column} \\
Y_{res} &= \text{number of pixels per row}
\end{aligned}
$$

We assume a left-handed coordinate system. Based on the above inputs, the location of the i, j^{th} pixel can be written as:

$$P_1 = (x, y, z) = (x, y, d) = \left(-\frac{W}{2} + \frac{W \cdot i}{X_{res} - 1}, \frac{H}{2} - \frac{H \cdot j}{Y_{res} - 1}, d\right)$$

The direction of the ray is:

$$dir = \frac{P_1 - P_0}{||P_1 - P_0||} = \frac{P_1}{||P_1||}$$

and the ray is

$$P = P_0 + t\ dir = t\ dir$$

2.2 Computing View Coordinate Axes: General Case

What if x, y, and z are not aligned with u, v, and n? Instead, assume we are given the following information:

$$P_0 \quad = \quad \text{location of camera}$$

$$
\begin{aligned}
\text{VPN} &= \text{normal to view plane} \\
\text{VUP} &= \text{up direction} \\
d &= \text{distance of camera from view screen} \\
H &= \text{height of screen} \\
W &= \text{width of screen} \\
X_{res} &= \text{number of pixels per column} \\
Y_{res} &= \text{number of pixels per row}
\end{aligned}
$$

We can compute unit vectors along the view coordinate axes, where here we are using a left-handed coordinate system.

$$
\begin{aligned}
\hat{n} &= \frac{\text{VPN}}{||\text{VPN}||} \\
\hat{u} &= \frac{\text{VUP} \times \text{VPN}}{||\text{VUP} \times \text{VPN}||} \\
\hat{v} &= \hat{n} \times \hat{u}
\end{aligned}
$$

Thus, in world coordinates, the difference $P_1 - P_0$ can be written as

$$P_1 - P_0 = \alpha\hat{u} + \beta\hat{v} + d\hat{n}$$

for some α and β. To determine α and β we assume that pixels are spread evenly over the screen and then convert from pixel (i, j) to a location on the screen as follows

$$(\alpha, \beta) = \left(-\frac{W}{2} + \frac{W \cdot i}{X_{res} - 1}, \frac{H}{2} - \frac{H \cdot j}{Y_{res} - 1}\right)$$

2.2.1 Example

Suppose $P_0 = \begin{pmatrix} -2 \\ -2 \\ 0 \end{pmatrix}$, $\text{VPN} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$, $\text{VUP} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, $d = 1$, $W = 2$, $H = 2$, $X_{res} = 201$, and $Y_{res} = 201$. Then

$$
\begin{aligned}
\hat{n} &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \\
\hat{u} &= \frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \\
\hat{v} &= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
\alpha &= -1 + \frac{i}{100}
\end{aligned}
$$

$$
\begin{aligned}
\beta &= 1 - \frac{j}{100} \\
dir &= P_1 - P_0 \\
P_1 - P_0 &= \alpha\hat{u} + \beta\hat{v} + d\hat{n} \\
P_1 - P_0 &= (-1 + \frac{i}{100})\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} + (1 - \frac{j}{100})\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + d\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 2 - \frac{i}{\sqrt{2}100} \\ \frac{1}{\sqrt{2}}(\frac{i}{100}) \\ 1 - \frac{j}{100} \end{pmatrix}
\end{aligned}
$$

For pixel $(i, j) = (10, 20)$,

$$
dir = P_1 - P_0 = \begin{pmatrix} \frac{1}{\sqrt{2}(2-\frac{1}{10})} \\ \frac{1}{10\sqrt{2}} \\ \frac{4}{5} \end{pmatrix}
$$

where we can then compute the ray as a function of the parameter t as follows

$$
P = P_0 + t\ dir = \begin{pmatrix} -2 \\ -2 \\ 0 \end{pmatrix} + t\begin{pmatrix} \frac{1}{\sqrt{2}(2-\frac{1}{10})} \\ \frac{1}{10\sqrt{2}} \\ \frac{4}{5} \end{pmatrix}
$$

3 Computing Intersections

3.1 Spheres

A sphere can be represented by it center $P_c = (a, b, c)$ and radius r. Given these, the equation for the sphere can be written as

$$
(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2.
$$

In vector notation, this becomes

$$
(P - P_c) \cdot (P - P_c) = r^2.
$$

To find the ray-sphere intersection we just insert the equation of ray in for P to give

$$
(P_0 + t\ dir - P_c) \cdot (P_0 + t\ dir - P_c) = r^2
$$

and solve for t. Multiplying out we obtain

$$
(P_0 - P_c) \cdot (P_0 - P_c) + 2\ dir \cdot (P_0 - P_c)t + \ dir \cdot dir\ t^2 = r^2
$$

which is a quadratic equation in t. Defining the scalars B and C as

$$\begin{aligned} B &= 2\ dir \cdot (P_0 - P_c) \\ C &= (P_0 - P_c) \cdot (P_0 - P_c) - r^2 \end{aligned}$$

and noting that $dir \cdot\ dir = 1$, we obtain the equation $t^2 + Bt + C = 0$. The solution is simply

$$t = \frac{-B \pm \sqrt{B^2 - 4c}}{2}.$$

There are 0, 1, or 2 real solutions to this depending on the value of the discriminant $D = \sqrt{B^2 - 4C}$. In particular,

$D < 0$	no intersections.
$D = 0$	1 intersection, ray grazes sphere.
$D > 0$	2 intersections, take smaller t value.

And if $t < 0$ then the object is behind viewer.

3.1.1 Example

Let the sphere be defined by: $P_c = \begin{pmatrix} 3 \\ 3 \\ 1 \end{pmatrix}$, $r = 2$

and the ray by $P_0 = \begin{pmatrix} 0 \\ -2 \\ 0 \end{pmatrix}$, $P_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$. (Use a left handed coordinate system.)

Then we find that

$$\begin{aligned} P_0 - P_c &= \begin{pmatrix} -3 \\ -5 \\ -1 \end{pmatrix} \\ dir &= \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix},\ B = \frac{-28}{\sqrt{6}} = -11.43,\ C = 31 \end{aligned}$$

to give

$$\begin{aligned} 0 &= t^2 - \frac{28}{\sqrt{6}} t + 31 \\ D &= B^2 - 4C = 6.67 \\ t &= \frac{-B \pm \sqrt{D}}{2} \\ &= 7.01(+) \text{ and } 4.42(-) \end{aligned}$$

So the intersection point is

$$P = P_0 + t\ dir = \begin{pmatrix} 1.80 \\ 1.61 \\ 1.80 \end{pmatrix}$$

3.2 Planes

A plane is defined by its normal N and a point on the plane Q_0. Recall that the equation of the plane is then given by

$$(P - Q_0) \cdot N = 0$$

To find the intersection of the plane and the ray we insert the expression for the ray into the above equation to obtain

$$(P_0 + t\ dir - Q_0) \cdot N = 0$$

and solving for t gives

$$t = \frac{(Q_0 - P_0) \cdot N}{dir \cdot N}$$

Note, a zero denominator means that the line and plane are parallel. In such a case there are either no intersections if the ray does not lie in the plane, or there are an infinite number of intersections.

Boxes aligned with the axes are also easy because they are just planes with the range constained.

4 Computing Pixel Color: Phong Lighting Model

At this point, we assume we have:

1. computed the equation of the ray (P_0 and dir) from camera to given pixel
2. identified the closest object that intersects the ray
3. computed the location of the intersection point (P) of the ray with the object
4. computed the surface normal at the intersection point
5. identified the location of all lights in the scene L_i.

The color of the given pixel at the computed intersection point is composed of 1) local color based on the interaction of surface with the ambient light and light coming directly from the light sources, 2) the color resulting from reflected light, and 3) color from refracted light. Here, we will discuss the local color as modeled by the Phong Lighting Model. Note that the reflected and refracted light can be computed by recursively applying Phong Model. The final pixel color is simply the sum of the local, reflected, and refracted contributions.

$$\text{pixel color} = \text{local color} + \alpha_1\text{*reflected} + \alpha_2\text{*refracted}$$

where α_1 and α_2 are material dependent weights (less than 1) that control how much light is reflected or refracted.

The Phong Model computes the local color by dividing the light-surface interaction into three components: ambient, diffuse, and specular. The local color is the sum of each contribution:

$$\text{local color} = \text{ambient} + \text{diffuse} + \text{specular}$$

4.1 Ambient Color Contribution

Ambient light is the "background" light that results from many many rays from all possible light sources bouncing off of objects multiple times. It is ambient light that allows us to see all surfaces in a room even if there is no direct light on the surfaces. Since it would be too complex to model every single ray, we instead model the average behavior. The modeling is very simplistic in that we assume that ambient light is constant over the entire scene. Although ambient light is a result of light sources in the room we model it as independent of any light source and only dependent on the scene.

Because ambient light is constant, it illuminates all surfaces with an equal brightness making objects monochrome.

Contribution of ambient light to the pixel color is:

$$C_a = k_a(c_r I_{a,r}, c_g I_{a,g}, c_b I_{a,b})$$

where

- k_a = ambient reflection coefficient, $0 \leq k_a \leq 1$
  k_a is a characteristic of the object, depending on the material make-up of the object.
- $c = (c_r, c_g, c_b)$ = color of the object which is also a characteristic of the object, depending on the material make-up of the object. The range for each component is either $0 \leq c_{r,g,b} \leq 1$. Sometimes k_a and c are combined. However, it is useful to keep them separate so that one can adjust the ambient coefficent while maintaining the same balance of color.
- $I_a = (I_{a,r}, I_{a,g}, I_{a,b})$ = the intensity of the ambient light. I_a is the same for all objects in the scene.
- Note, the final pixel color should never exceed 1.

4.2 Diffuse and Specular

Diffuse and specular components are a result of light coming directly from a light source to the intersection point and then being reflected from an object back to the

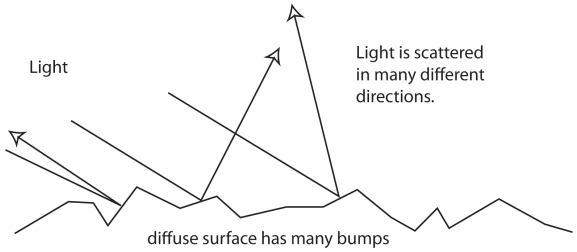
observer's eyes. Surfaces (i.e. materials) that are rough tend to scatter the light in all directions. Such materials tend to have a rather dull finish and the illumination is referred to as diffuse. While materials that are smooth and highly reflective (i.e. shiny) tend to reflect light very non-uniformly. Such objects tend to have what are called specular highlights. While the separation between these two is rather artificial, it is still useful to model the diffuse and specular components as being separate.

For diffuse and specular we consider light that hits the intersection point directly from the light. We want to compute how much light from the light source that hits the intersection actually makes it back to the eye.

The ray that goes from the light to the object is called the **shadow ray**. Note that if the shadow ray hits an opaque object before reaching the intersection point then that light does not contribute to the light source at the intersection point. need picture.

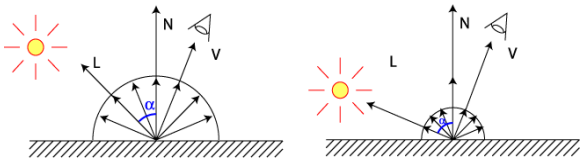
4.2.1 Diffuse Light Component

Objects that have a dull, matte finish are said to have a large diffuse component. The surface at the macroscopic level may appear smooth but at the microscopic level, there are many tiny bumps that scatter light in all directions.



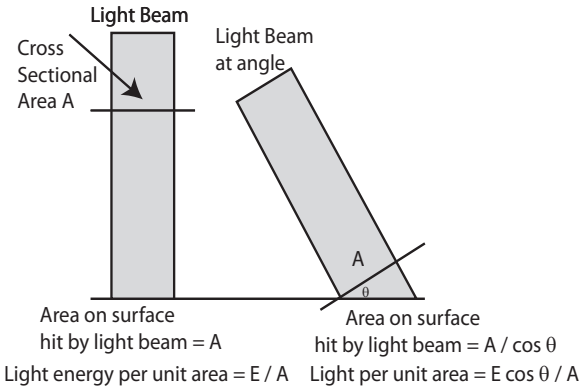
The surface normal at any given point, is generally not aligned with the normal of the macroscopic object. As a result, light is scattered in many directions. We make the simplifying assumption that the light is scattered *equally* in all directions so that the brightness is independent of viewing angle.

The brightness,however, does depend on the angle that the light ray makes with the surface normal.



The reason for this is as follows. Imagine a cylinder of light coming from the light source and falling on the object (here we are ignoring light attenuation). The cross sectional area of this cylinder is A. The energy per unit area hitting the surface when the light is right overhead is E/A where E is the energy emitted from the light source. If the light is at an angle then the surface area hit by the light is larger and is given by $A_\theta = A/\cos\theta$ so the energy per unit area is $E\cos\theta/A$, i.e. it is smaller by an amount $\cos\theta$. Thus the brightness is a function of $L \cdot N = \cos\theta$ where L is the

unit vector pointing from the intersection point to the light source and N is the unit normal to the surface.



Why does the brightness not depend on the viewing angle? Once the light is fixed, we have a fixed energy per unit area hitting the surface. As the viewing angle increases the area viewed increases. However, the amount of this energy that gets reflected decreases so as to offset the increase in viewing area. Thus the view sees the same brightness from any angle.

For a monochrome light source, the diffuse component is given by

$$C_d = k_d \sum_j I_j (L_j \cdot N)$$

where

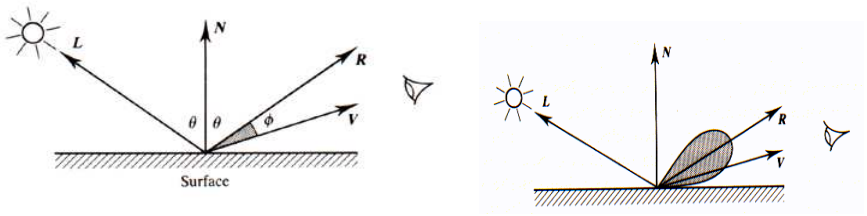
- k_d = coefficient of diffuse reflection of the object being intersected
- N = unit normal at intersection point
- L_j = unit vector pointing from the intersection point to the j^{th} light source.
- I_j = the intensity of the j^{th} light source.
- Note: if $L_j \cdot N < 0$ and the intersected object is opaque then the diffuse component for that light is zero because the light is behind the object.

If we add color, then the intensity becomes $I_j = (I_{j,r}, I_{j,g}, I_{j,b})$ and we introduce the diffuse color of the object $c = (c_r, c_g, c_b)$ which may or may not be different from the ambient color. The diffuse component then becomes

$$C_d = k_d \sum_j (c_r I_{j,r}, c_g I_{j,g}, c_b I_{j,b})(L_j \cdot N)$$

4.2.2 Specular Light Component

Objects that are shiny have what we call specular highlights. Unlike diffuse surfaces, shiny surfaces are very smooth even at the microscopic level. When light hits the surface, most of the light is reflected only in one direction, the angle of reflection.



For a monochrome light source, the specular component is given by

$$C_s = k_s \sum_j I_j (V \cdot R_j)^n$$

where

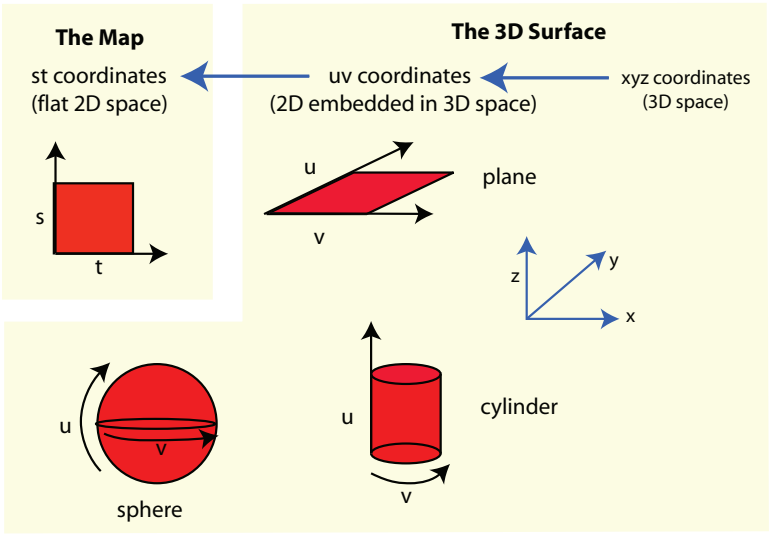
- k_s = coefficient of specular reflection of the object being intersected
- R_j = unit direction of the reflected ray of the j^{th} light source.
- V = unit direction of the viewer
- I_j = the intensity of the j^{th} light source.
- n = specularity (measure of the sharpness of the highlight). Can be in the range from 1 to several hundred.
- Note: the color of the specular highlight is generally set to the color of the light source and not the object.

Texture Types

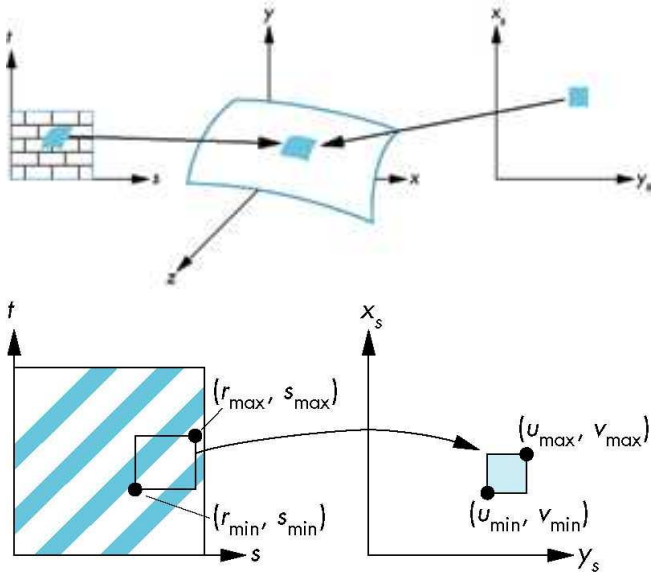
- The texture data can be a 2D image or a 3D solid texture.
- The coordinates of the texture are called the *texture parameters* and the assignment of the texture parameters to the surface is called *parameterization*. Texture parameters are commonly denoted u, v for 2D and u, v, w for 3D.
- Texture images, *texture(u,v)* are good when the surface has a natural parameterization and/or you want a "decal" or "projected" look. This is the most common type of texture mapping.
- Solid textures, texture(u,v,w), are good when you want a surface to appear carved out of a material (e.g. wood, marble).
- Textures can be stored as a raster image, a 3D discrete grid, or can be computed on the fly procedurally. In procedural mapping (surface or solid): the "image" is defined mathematically.

Texture Parameterizationi: 2D Image Textures

2D texture is mapped to a 2D surface embedded in 3D space. See figure below. In this process we assume that the image is stored in some nxn array. Each point in the array is called a texel. WLOG, we can scale this so that the image fits in a range $0 \leq u, v \leq 1$.



Each pixel on the screen represents some portion of the (x,y,z) surface. This surface then gets mapped to 2D uv space, which is then mapped to the image (st space) to determine the color. Aliasing can be a problem depending on how the size of a pixel in screen space compares to the area the pixel is covering in image (st) space.

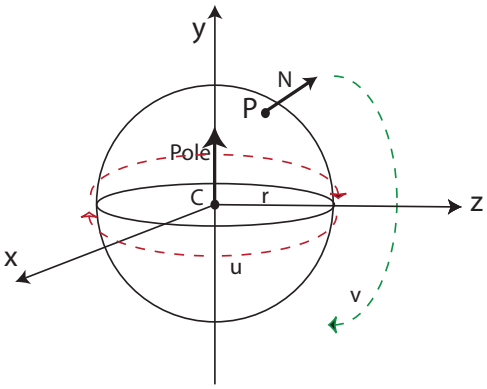


Finding a reasonable mapping between uv and xyz can be a particular problem if the surface is not flat. Also, edges may need to be matched up.

Parameterization is somewhat straight forward for spheres, planes, cylinders,and rectangular surfaces (see sphere below).

Parameterization is somewhat less obvious for arbitrary polygons or volumes. In the case of volumes, one can do a 2-step process of first mapping to an easy to parameterize intermediate shape such as a sphere or cylinder. The second step is to map from the intermediate shape to the surface of the actual object.

Spheres



Assume center is at C and radius $= r$.

We use a left handed coordinate system. v is zero at north pole and 1 at south pole. The angle that is made is called θ which ranges between 0 and π.

u moves around in the xz plane around y (use left-hand rule to get direction). The angle is ϕ which ranges between 0 and 2π.

Let P be the intersection point. Then the normal at P is

$$N = \frac{P - C}{||P - C||} = \frac{P - C}{r}$$

1. Compute v:

$$v = \frac{\theta}{\pi}$$

But $cos\theta = N \cdot Pole$ so that

$$v = \frac{cos^{-1}(N \cdot Pole)}{\pi}$$

Letting $Pole = (0, 1, 0)$ then

$$v = \frac{cos^{-1}((p_y - C_y)/r)}{\pi}$$

2. Compute u: Note, if $v = 0$ or 1 then u can be anything.

$$u = \frac{\phi}{2\pi}$$

But $\tan\phi = x/z = \frac{p_x - C_x}{p_z - C_z}$ so that

$$u = 0.5 + \frac{1}{2\pi}\tan^{-1}\frac{p_x - C_x}{p_z - C_z}$$

Note, we add .5 because $\tan^{-1}$ returns a value between $-\pi$ and π rather than 0 and 2π .

3D Parameterization: 3D Solid textures

(u,v,w) are usually taken to be the world space or object space. Often these are defined procedurally (wood, marble)

Bump Mapping

Bump mapping: make small perturbations to the surface normal so that smooth objects appear to look bumpy, rough, wrinkled, or rippled.