# Chapter 12

# Implicit and Parametric Surfaces

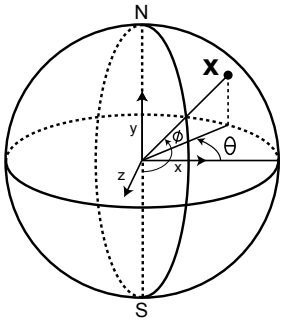
In this chapter we will look at the two ways of mathematically specifying a geometric object. You are probably already familiar with two ways of representing a sphere of radius $r$ and center at the origin. The first is the *implicit* form

$$x^2 + y^2 + z^2 = r^2 \quad \text{or} \quad x^2 + y^2 + z^2 - r^2 = 0. \tag{12.1}$$

The second is the *parametric* form expressed in spherical coordinates

$$\begin{aligned} x &= r \sin\phi \cos\theta, \\ y &= -r \cos\phi, \\ z &= -r \sin\phi \sin\theta, \end{aligned}$$

where $0 \leq \theta \leq 2\pi$ is an angle parameter for longitude measured from the positive $x$ axis as a positive rotation about the $y$ axis, and $0 \leq \phi \leq \pi$ is an angle parameter for latitude measured from the negative $y$ axis as a positive rotation about the $z$ axis.



## 12.1 Implicit representations of surfaces

An implicit representation takes the form $F(\mathbf{x}) = 0$ (for example $x^2 + y^2 + z^2 - r^2 = 0$), where $\mathbf{x}$ is a point on the surface implicitly described by the function $F$. In other words, point $\mathbf{x}$ is on the surface if and only if the relationship $F(\mathbf{x}) = 0$ holds for $\mathbf{x}$. With this representation, we can easily test a given point $\mathbf{x}$ to see if it is on the surface, simply by checking the value of $F(\mathbf{x})$. However, if you examine this representation, you will see that there is no direct way that we can systematically generate consecutive points on the surface. This is why the representation is called implicit; it provides a test for determining whether or not a point is on the surface, but does not give any explicit rules for generating such points.

Usually, an implicit representation is constructed so that it has the property that

$$\begin{aligned} F(\mathbf{x}) &> 0, \text{ if } \mathbf{x} \text{ is "above" the surface,} \\ F(\mathbf{x}) &= 0, \text{ if } \mathbf{x} \text{ is exactly on the surface,} \\ F(\mathbf{x}) &< 0, \text{ if } \mathbf{x} \text{ is "below" the surface.} \end{aligned}$$

For example, this property holds for the sphere given by Equation 12.1. Rewriting this equation in vector form we have

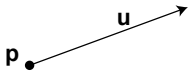
$$\mathbf{x}^2 - r^2 = 0 \quad (\text{where } \mathbf{x}^2 = \mathbf{x} \cdot \mathbf{x} = x^2 + y^2 + z^2).$$

You can see that when $\|\mathbf{x}\| > r$, then $\mathbf{x}^2 - r^2 > 0$, indicating that the point is outside the sphere. Likewise, when $\|\mathbf{x}\| < r$, then $\mathbf{x}^2 - r^2 < 0$, indicating that the point is inside the sphere.

*Thus, an implicit representation allows for a quick and easy inside-outside (or above-below) test.*

Implicit representations are also ideal for raytracing. Given a ray expressed as a starting point $\mathbf{p}$ and a direction vector $\mathbf{u}$, all points on the ray can be expressed in parametric form as

$$\mathbf{x}(t) = \mathbf{p} + t\mathbf{u},$$




where $t$ is the ray parameter, measuring distance along the ray. Inserting this parametric ray equation into the implicit representation gives

$$F[\mathbf{x}(t)] = 0,$$

which can often be conveniently solved for $t$.

We have seen this for ray-plane intersection. The implicit equation for a plane is

$$ax + by + cz + d = 0,$$

or letting $\mathbf{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} / \sqrt{a^2+b^2+c^2}$, and $e = d/\sqrt{a^2+b^2+c^2}$, in vector form we have $\mathbf{n}\cdot\mathbf{x}+e = 0$. Inserting the ray equation for $\mathbf{x}$, we have $\mathbf{n}\cdot(\mathbf{p}+t\mathbf{u})+e = 0$ and solving for $t$, we have

$$t = -\frac{e + \mathbf{n}\cdot\mathbf{p}}{\mathbf{n}\cdot\mathbf{u}},$$

which is the distance, along the ray, of the ray-plane intersection.

Similarly for the sphere, we have the implicit form

$$\mathbf{x}^2 - r^2 = 0.$$

Inserting the ray equation for $\mathbf{x}$ gives

$$(\mathbf{p} + t\mathbf{u})^2 - r^2 = 0,$$

or

$$t^2\mathbf{u}^2 + 2t\mathbf{p}\cdot\mathbf{u} + \mathbf{p}^2 - r^2 = 0.$$

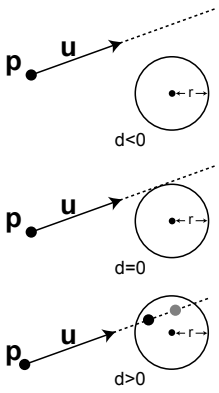
Since $\mathbf{u}$ is a unit vector, $\mathbf{u}^2 = 1$, and this reduces to

$$t^2 + 2t\mathbf{p}\cdot\mathbf{u} + \mathbf{p}^2 - r^2 = 0,$$

which by the quadratic formula has solutions

$$t = -\mathbf{p}\cdot\mathbf{u} \pm \sqrt{(\mathbf{p}\cdot\mathbf{u})^2 - \mathbf{p}^2 + r^2}.$$

If the discriminant $d = (\mathbf{p}\cdot\mathbf{u})^2 - \mathbf{p}^2 + r^2 < 0$ we have no real solutions and the ray misses the sphere. If $d = 0$ we have one solution $t = -\mathbf{p}\cdot\mathbf{u}$, so the ray must be exactly tangent to the sphere, and if $d > 0$ we have two solutions, one where the ray enters the sphere and the other where it leaves the sphere. The figure to the right shows the cases.




*So, an implicit representation can work very well in a raytracer.*

## 12.2 Parametric representations of surfaces

A parametric surface representation has other attributes that make it useful in graphics. This representation, speaking generally, can be written $\mathbf{x} = F(u, v)$, where $u$ and $v$ are surface parameters, and $\mathbf{x}$ is a point on the surface. The parameters are often chosen so that on the surface being described, $0 \le u, v \le 1$. For example, in the parametric representation of the sphere introduced at the beginning of this chapter, we can normalize the two angle parameters, letting $u = \theta/2\pi$, and $v = \phi/\pi$.

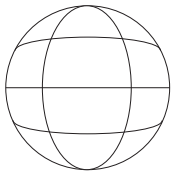
Unlike an implicit representation of a surface, a parametric representation allows us to directly generate points on the surface. All that is required is to choose values of the parameters $u$ and $v$ and then $\mathbf{x}(u, v) = F(u, v)$. If this is done in a systematic way over the range of possible $u$ and $v$ values, it is possible to generate a set of points sampling the entire surface. However, a parametric surface is difficult to raytrace, since there is no direct way to take an arbitrary point in space and test to see if it is on the surface.

Because a parametric representation allows us to systematically generate points on a surface, it is the ideal form if we want to be able to generate a polygonal surface approximating the mathematical surface. This is exactly what we want in applications utilizing graphics hardware and graphics APIs like *OpenGL* and *DirectX* to render a scene. These require a polygonal representation of the geometry, since their architecture is designed to efficiently process triangles. Producing a polygonal surface simply requires iterating over the range of the two parameters $u$ and $v$, as in the psudocode below, which shows how to generate all of a model's vertices for a chosen resolution:

```
m = number of steps in v;
n = number of steps in u;
Δv = 1.0 / m; Δu = 1.0 / n;
for(i = 0; i < m; i++){
  v = i * Δv;
  for(j = 0; j < n; j++){
    u = j * Δu;
    x = F(u, v);
    insertvertex(x);
  }
}
```

In the pseudocode, the call to `insertvertex(x)` adds vertex `x` to a table of vertices for the model. A similar loop could then be used to generate all of the polygonal faces from the vertices.

For example, let us consider the parametric representation of the sphere. The figure to the right shows how we might generate polygons to tile a sphere. The tiling has triangular faces at the poles, but quadrilateral faces away from the poles. All of the triangles at one pole share a common vertex. Thus, the code to generate a sphere, including surface normals and texture coordinates at a vertex might be as follows:



sphere: $\mathtt{m} = 4$, $\mathtt{n} = 6$, $\Delta u = 1/6$, $\Delta v = 1/4$

```
Δv = 1.0 / m; Δu = 1.0 / n;
// create all of the vertices
insertvertex((0, −r, 0)); // south pole, vertex 0
insertvertex((0, r, 0)); // north pole, vertex 1
for(i = 1; i < m − 1; i++){ // iterate over latitudes
  v = i * Δv; φ = πv;
  for(j = 0; j < n; j++){ // iterate over longitudes
    u = j * Δu; θ = 2πu;
    x = [ r sin φ cos θ ; −r cos φ ; −r sin φ sin θ ];
    insertvertex(x); // vertex i + 1 + n * j
    insertnormal(normalize(x));
    inserttexcoords(u, v);
  }
}
// create triangular faces at the two poles
for(j = 0; j < n − 1; j++){
  insertface(3, 0, j+3, j+2);
  insertface(3, 1, (m−2)*n + j+2, (m−2)*n+j+3);
}
// create quadrilateral faces
for(i = 1; i < m − 2; i++)
  for(j = 0; j < n − 1; j++)
    insertface(4, (i−1)*n+j+2, (i−1)*n+j+3, i*n+j+3, i*n+j+2);
```

Note, that in the above we are using the convention that the call
`insertface(` *number of vertices in face, list of vertex indices*)
adds a list of vertex indices to a table of faces describing the sphere.

*Thus, we see that a parametric representation is ideal if one wants to create an approximating polygonal model from a surface.*

Another advantage of a parametric surface, is that since the surface is explicitly parameterized, for every point on the surface, we have parametric coordinates $(u, v) = F^{-1}(\mathbf{x})$, so if the inverse of $F$ is easily computable, we have a pair of parameters for each surface point that can be used to index a texture map for coloring the surface. For example for our parametric sphere example,

$$\frac{-z}{x} = \frac{r \sin\phi \sin\theta}{r \sin\phi \cos\theta} = \tan\theta,$$

and

$$\frac{y}{-r} = \cos\theta,$$

so that

$$\theta = \tan^{-1}\frac{-z}{x},\ \phi = \cos^{-1}\frac{y}{-r},$$

and

$$u = \frac{1}{2\pi}\tan^{-1}\frac{-z}{x},\ v = \frac{1}{\pi}\cos^{-1}\frac{y}{-r}.$$

In practice, we often have $u$ and $v$ available at the time when we generate a polygonal surface, so we do not actually need to know the inverse function, since we can just store the $(u, v)$ coordinates of a point in a data structure along with the position of the point.

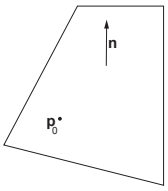
*So, a parametric representation can work very well when texture mapping is being used.*

## 12.3 Implicit and parametric plane representations

Our implicit definition of a plane, in vector form, is given by

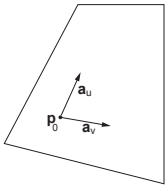
$$\mathbf{n} \cdot \mathbf{x} - \mathbf{n} \cdot \mathbf{p}_0 = 0,$$

where $\mathbf{n}$ is the unit surface normal of the plane and $\mathbf{p}_0$ is any point known to be on the plane. In this case, the ray intersection with the plane is given by




$$t = \frac{(\mathbf{p}_0 - \mathbf{p}) \cdot \mathbf{n}}{\mathbf{u} \cdot \mathbf{n}} \text{ for ray } \mathbf{x} = \mathbf{p} + t\mathbf{u}.$$

We can also create a parametric representation of the plane and use this to tile the plane with polygons. What we need to do this is to create a coordinate frame on the plane. Thus, we need two non-parallel vectors we will call $\mathbf{a}_u$ and $\mathbf{a}_v$ and a point $\mathbf{p}_0$, on the plane, that will serve as the origin. If $\mathbf{a}_u$ and $\mathbf{a}_v$ are orthogonal unit vectors, we have an orthogonal coordinate system on the plane. If we then apply some distance measure, for example $w$ =width in the $\mathbf{a}_u$ direction and $h$ = height in the $\mathbf{a}_v$ direction, any point $\mathbf{x}$ on the plane is given by




$$\mathbf{x} = \mathbf{p}_0 + wu\mathbf{a}_u + hv\mathbf{a}_v.$$

Now, if $u$ and $v$ are made to vary from 0 to 1 in a nested loop, we can generate a set of quadrilaterals forming a rectangle of width $w$ and height $h$ on the plane. These quadrilaterals can be easily subdivided to create triangles if we need them.