Some Mathematical Elements of Graphics

Written October 1999, modifications made September 2000 and October 2002.

1 Ray tracing primitives

1.1 Mathematical preliminaries

1.1.1 Vector arithmetic

It is helpful to remember your vector arithmetic. A 3D vector is represented thus:

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

For ease of writing such definitions in text we may say $\mathbf{P} = (x, y, z)$, where we understand that this ordered triple is equivalent to the vector.
The magnitude of vector $\mathbf{P}$ is:

$$|\mathbf{P}| = \sqrt{x^2 + y^2 + z^2} \tag{2}$$

The dot product of two vectors ($\mathbf{A} = (x_A, y_A, z_A)$ and $\mathbf{B} = (x_B, y_B, z_B)$) is:

$$\mathbf{A} \cdot \mathbf{B} = x_A x_B + y_A y_B + z_A z_B \tag{3}$$

which could also be written as a matrix multiplication:

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{A}^T \mathbf{B} = [x_A y_A z_A] \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{4}$$

The dot product is a scalar value equal to:

$$\mathbf{A} \cdot \mathbf{B} = |A||B| \cos\theta \tag{5}$$

where θ is the angle between the two vectors. Note that this means that:

$$\mathbf{P} \cdot \mathbf{P} = |\mathbf{P}|^2 \tag{6}$$

and hence:

$$|\mathbf{P}| = \sqrt{\mathbf{P} \cdot \mathbf{P}} \tag{7}$$

Also note that the dot product between two mutually perpendicular vectors is always zero.
The cross product (vector product) of these two vectors is:

$$\mathbf{A} \times \mathbf{B} = \begin{bmatrix} y_A z_B - z_A y_B \\ z_A x_B - x_A z_B \\ x_A y_B - y_A x_B \end{bmatrix} \tag{8}$$

The cross product is a vector which is perpendicular to both $\mathbf{A}$ and $\mathbf{B}$. This is a very handy way to make a new vector which is guaranteed perpendicular to a given vector. It also means that:

$$\begin{aligned} \mathbf{A} \cdot (\mathbf{A} \times \mathbf{B}) &= 0 && (9) \\ \mathbf{B} \cdot (\mathbf{A} \times \mathbf{B}) &= 0 && (10) \end{aligned}$$

though this may be getting a bit esoteric and so let's move on.

1.1.2 Points and vectors

Both points and vectors are three-tuples of real numbers. However, they are different beasts and must be treated differently. They point is an absolute position in space relative to some fixed origin, $\mathbf{O} = (0, 0, 0)$. A vector is an offset, specifying a distance and direction but *not* an absolute position.

Only certain arithmetic operations are permitted on points and vectors. Let $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \ldots$ represent points and $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3, \ldots$ represent vectors. Vectors can be added,

$$\mathbf{D}_3 = \mathbf{D}_1 + \mathbf{D}_2, \tag{11}$$

but points cannot be added. The difference of two vectors is a vector,

$$\mathbf{D}_3 = \mathbf{D}_1 - \mathbf{D}_2, \tag{12}$$

but the difference of two points is also a vector:

$$\mathbf{D}_1 = \mathbf{P}_1 - \mathbf{P}_2. \tag{13}$$

You can add a vector to a vector,

$$\mathbf{D}_3 = \mathbf{D}_1 + \mathbf{D}_2, \tag{14}$$

or a vector to a point,

$$\mathbf{P}_2 = \mathbf{P}_1 + \mathbf{D}_1, \tag{15}$$

but *not* a point to a point. It is possible to take a weighted average of points,

$$\mathbf{P}_{\text{new}} = \alpha_1 \mathbf{P}_1 + \alpha_2 \mathbf{P}_2 + \cdots + \alpha_k \mathbf{P}_k, \sum_{i=1}^{k} \alpha_i = 1 \tag{16}$$

provided that the weights sum to one, otherwise the operation makes no sense.

If you transform your coordinate system, then all points scale, rotate, and translate as you would expect. Vectors also scale and rotate but they do not translate. To see why this is the case, consider $\mathbf{D}_1 = \mathbf{P}_1 - \mathbf{P}_2$. Let the two points be translated: $\mathbf{P}'_1 = \mathbf{P}_1 + \mathbf{D}_o$ and $\mathbf{P}'_2 = \mathbf{P}_2 + \mathbf{D}_o$. Then:

$$\begin{aligned} \mathbf{D}'_1 &= \mathbf{P}'_1 - \mathbf{P}'_2 && (17) \\ &= (\mathbf{P}_1 + \mathbf{D}_o) - (\mathbf{P}_2 + \mathbf{D}_o) && (18) \\ &= (\mathbf{P}_1 - \mathbf{P}_2) + (\mathbf{D}_o - \mathbf{D}_o) && (19) \\ &= (\mathbf{P}_1 - \mathbf{P}_2) && (20) \\ &= \mathbf{D}_1 && (21) \end{aligned}$$

1.2 Equation of a ray

A ray is defined by an origin or eye point, $\mathbf{E} = (x_E, y_E, z_E)$, and an offset vector, $\mathbf{D} = (x_D, y_D, z_D)$. The equation for the ray is:

$$\mathbf{P}(t) = \mathbf{E} + t\mathbf{D}, t \geq 0 \tag{22}$$

This is obviously equivalent to the three equations:

$$\left.\begin{array}{l} x(t) = x_E + tx_D \\ y(t) = y_E + ty_D \\ z(t) = z_E + tz_D \end{array}\right\} t \geq 0 \tag{23}$$

When finding a ray-object intersection point, we are looking for the intersection point with the *lowest non-negative* value of t.

1.3 Equations for the primitives

1.3.1 Sphere

The unit sphere, centred at the origin, has the implicit equation:

$$x^2 + y^2 + z^2 = 1 \tag{24}$$

In vector arithmetic, this becomes:

$$\mathbf{P} \cdot \mathbf{P} = 1 \tag{25}$$

To find the intersection between this sphere and an arbitrary ray, substitute the ray equation (Equation 23) in the sphere equation (Equation 24):

$$\begin{aligned} & (x_E + tx_D)^2 + (y_E + ty_D)^2 + (z_E + tz_D)^2 = 1 && (26) \\ \Rightarrow\quad & t^2(x_D^2 + y_D^2 + z_D^2) + t(2x_Ex_D + 2y_Ey_D + 2z_Ez_D) && \\ & +(x_E^2 + y_E^2 + z_E^2 - 1) = 0 && (27) \\ \Rightarrow\quad & at^2 + bt + c = 0 && (28) \\ \Rightarrow\quad & t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} && (29) \end{aligned}$$

where $a = x_D^2 + y_D^2 + z_D^2$, $b = 2x_Ex_D + 2y_Ey_D + 2z_Ez_D$, and $c = x_E^2 + y_E^2 + z_E^2 - 1$. This gives zero, one, or two real values for t. If there are zero real values then there is no intersection between the ray and the sphere. If there are either one or two real values then chose the *smallest, non-negative* value, as the intersection point. If there is no non-negative value, then the line (of which the ray is a part) *does* intersect the sphere, but the intersection point is not on the part of the line which constitutes the ray. In this case there is again no intersection point between the ray and the sphere.

An alternative formulation is to use the vector versions of the equations (Equations 22 and 25):

$$\begin{aligned} & (\mathbf{E} + t\mathbf{D}) \cdot (\mathbf{E} + t\mathbf{D}) = 1 && (30) \\ \Rightarrow\quad & t^2(\mathbf{D} \cdot \mathbf{D}) + t(2\mathbf{E} \cdot \mathbf{D}) + (\mathbf{E} \cdot \mathbf{E} - 1) = 0 && (31) \\ \Rightarrow\quad & at^2 + bt + c = 0 && (32) \\ \Rightarrow\quad & t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} && (33) \end{aligned}$$

Where $a = \mathbf{D} \cdot \mathbf{D}$, $b = 2\mathbf{E} \cdot \mathbf{D}$, and $c = \mathbf{E} \cdot \mathbf{E} - 1$. In other words, exactly the same result, expressed in a more compact way. *Graphics Gems I* (p. 388) describes yet another way of arriving at the same result.

You will remember, from Part IB, that we need to know the normal vector at the intersection point in order to calculate the illumination and/or find the reflection ray. For a sphere centred at the origin, the normal is a vector passing through the origin and the point of intersection.

$$\begin{aligned} \mathbf{N} &= (x, y, z) - (0, 0, 0) \quad (34) \\ &= (x, y, z) \quad (35) \end{aligned}$$

Remember that points and vectors are different beasts. Equation 34 is the subtraction of one point from another while Equation 35 is a vector. So (x, y, z) in Equation 34 is a point while (x, y, z) in Equation 35 is a vector.

If the origin of the ray lies inside the sphere then the normal at the intersection point is the negative of that in Equation 35, that is, for the case of **E** inside the sphere,

$$\mathbf{N} = (-x, -y, -z). \quad (36)$$

To check whether **E** is inside the unit sphere we simply need to check whether $|\mathbf{E}| < 1$.

1.3.2 Cylinder

The *infinite* unit cylinder aligned along the z-axis is defined as:

$$x^2 + y^2 = 1 \quad (37)$$

To intersect a ray with this, substitute Equation 23 in Equation 37.

$$\begin{aligned} & (x_E + t x_D)^2 + (y_E + t y_D)^2 = 1 \quad (38) \\ \Rightarrow\quad & t^2(x_D^2 + y_D^2) + t(2x_E x_D + 2y_E y_D) \\ & +(x_E^2 + y_E^2 - 1) = 0 \quad (39) \\ \Rightarrow\quad & at^2 + bt + c = 0 \quad (40) \\ \Rightarrow\quad & t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (41) \end{aligned}$$

where $a = x_D^2 + y_D^2$, $b = 2x_E x_D + 2y_E y_D$, and $c = x_E^2 + y_E^2 - 1$.

The *finite* open-ended unit cylinder aligned along the z-axis is defined as:

$$x^2 + y^2 = 1, z_{\min} < z < z_{\max} \quad (42)$$

The only difference between this and Equation 37 being the restriction on z. To handle this finite length cylinder, solve Equation 41 above. This gives, at most, two values of t. Call these t_1 and t_2. Calculate z_1 and z_2 using Equation 23 ($z_1 = z_E + t_1 z_D$ and $z_2 = z_E + t_2 z_D$) and then check $z_{\min} < z_1 < z_{\max}$ and $z_{\min} < z_2 < z_{\max}$. Whichever intersection point passes this test and, if both pass the test, has the smallest non-negative value of t, is the closest intersection point of the ray with the open-ended finite cylinder.

If we wish the finite length cylinder to be closed we must formulate an intersection calculation between the ray and the cylinder's end caps. The end caps have the formulae:

$$z = z_{\min}, \quad x^2 + y^2 \leq 1 \tag{43}$$
$$z = z_{\max}, \quad x^2 + y^2 \leq 1 \tag{44}$$

Once you have calculated the solutions to Equation 41 you will either know that there are no intersections with the infinite cylinder or you will know that there are one or two real intersection points (t_1 and t_2). The previous paragraph explained how to ascertain whether these correspond to points on the finite length open-ended cylinder. Now, if z_1 and z_2 lie either side of $z_{\min}$ we know that the ray intersects the $z_{\min}$ end cap, and can calculate the intersection point as:

$$t_3 = \frac{z_{\min} - z_E}{z_D} \tag{45}$$

A similar equation holds for the $z_{\max}$ end cap. Note that the ray may intersect both end caps, for example when $z_1 < z_{\min}$ and $z_2 > z_{\max}$. Also, note that a ray parallel to the z-axis is a special case and needs to be handled separately. It is left as an exercise how to check for and how to handle this case.

The normal vector of this cylinder, at intersection point (x, y, z) on the surface of the infinite cylinder, will be $(x, y, 0)$ if the ray's origin is outside the infinite cylinder and $(-x, -y, 0)$ if the ray's origin is inside. For a finite cylinder, if the intersection point is on an end cap, then the normal vector will be $(0, 0, -1)$ or $(0, 0, 1)$ depending on which end cap is hit and whether the origin of the ray is inside or outside the finite cylinder. It is an exercise for the reader to work out how to ascertain whether the ray's origin, **E**, is inside or outside of the cylinder in both the finite and infinite cases.

1.3.3 Cone

The *infinite* double cone[1] aligned along the z-axis and having unit slope is defined as:

$$x^2 + y^2 = z^2 \tag{46}$$

To intersect a ray with this, substitute Equation 23 in Equation 46.

$$\begin{aligned}
& (x_E + tx_D)^2 + (y_E + ty_D)^2 = (z_E + tz_D)^2 && (47)\\
\Rightarrow\quad & t^2(x_D^2 + y_D^2 - z_D^2) + t(2x_Ex_D + 2y_Ey_D - 2z_Ez_D) && \\
& +(x_E^2 + y_E^2 - z_E^2) = 0 && (48)\\
\Rightarrow\quad & at^2 + bt + c = 0 && (49)\\
\Rightarrow\quad & t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} && (50)
\end{aligned}$$

where $a = x_D^2 + y_D^2 - z_D^2$, $b = 2x_Ex_D + 2y_Ey_D - 2z_Ez_D$, and $c = x_E^2 + y_E^2 - z_E^2$.

The *finite* open-ended cone aligned along the z-axis is defined as:

$$x^2 + y^2 = z^2, z_{\min} < z < z_{\max} \tag{51}$$

[1]*double cone* means that it is two "standard" cones joined at their apices.

The only difference between this and Equation 46 being the restriction on z. Note that if $z_{\min}$ and $z_{\max}$ are both positive or both negative then you get a single cone with its top truncated. If either $z_{\min}$ or $z_{\max}$ is zero you get a single cone with its apex at the origin.

To handle this finite length cone you proceed as for the finite length cylinder, with several subtle modifications. It is left as an exercise to work out all the possible cases, how each should be handled, and how the intersection point and normal vector should be calculated in each case.

1.3.4 Torus

A torus is defined by two parameters: the radius of the torus (that is the radius of the torus's defining circle, measured from the origin) and the radius of the tube (the perpendicular distance from the defining circle to the surface of the torus). These are R and r respectively. Normally $R > r$.

An implicit definition of the torus is:

$$\left(\sqrt{x^2+y^2}-R\right)^2+z^2=r^2 \tag{52}$$

The torus can also be defined parametrically in terms of two angles, θ and ϕ, where θ can be thought of as the angle around the defining circle and ϕ the angle around the inside of the tube:

$$\begin{aligned} x &= (R+r\cos\phi)\cos\theta & (53)\\ y &= (R+r\cos\phi)\sin\theta & (54)\\ z &= r\sin\phi & (55)\end{aligned}$$

Substituting these three equations into Equation 52 will show that they are correct and is a useful exercise in algebraic manipulation.

To find the intersection points of a ray with a torus you need to substitute Equation 23 in Equation 52. Equation 56 is that substitution with the $(\sqrt{x^2+y^2}-R)^2$ term expanded, the resulting square root term placed on one side of the equals sign, and all other terms placed on the other side:

$$\begin{aligned} & 2R\sqrt{x_E^2+2tx_Ex_D+t^2x_D^2+y_E^2+2ty_Ey_D+t^2y_D^2} \\ = \; & R^2+x_E^2+2tx_Ex_D+t^2x_D^2+y_E^2+2ty_Ey_D+t^2y_D^2+z_E^2+2tz_Ez_D+t^2z_D^2-r^2 \end{aligned} \tag{56}$$

If we now square both sides we will get a quartic equation in t. This can be solved using a standard quartic root finder to find the four roots of the equation[2] (there are up to four intersection points between a torus and an arbitrary ray). A quartic root finder is described in *Graphics Gems V* (p. 3).

[2]The equation itself is rather fierce and you would not be expected to do the full expansion in an exam. For those who are interested, it looks like this:
$t^4(x_D^4+y_D^4+z_D^4+2x_D^2y_D^2+2x_D^2z_D^2+2y_D^2z_D^2)+t^3(4x_D^3x_E+4y_D^3y_E+4z_D^3z_E+4x_D^2y_Dy_E+4x_D^2z_Dz_E+4x_Dx_Ey_D^2+4y_D^2z_Dz_E+4x_Dx_Ez_D^2+4y_Dy_Ez_D^2)+t^2(-2R^2x_D^2-2R^2y_D^2+2R^2z_D^2-2r^2x_D^2-2r^2y_D^2-2r^2z_D^2+6x_D^2x_E^2+2x_E^2y_D^2+8x_Dx_Ey_Dy_E+2x_D^2y_E^2+6y_D^2y_E^2+2x_E^2z_D^2+2y_E^2z_D^2+8x_Dx_Ez_Dz_E+8y_Dy_Ez_Dz_E+2x_D^2z_E^2+2y_D^2z_E^2+6z_D^2z_E^2)+t(-4R^2x_Dx_E-4R^2y_Dy_E+4R^2z_Dz_E-4r^2x_Dx_E-4r^2y_Dy_E-4r^2z_Dz_E+4x_Dx_E^3+4x_E^2y_Dy_E+4x_Dx_Ey_E^2+4y_Dy_E^3+4x_E^2z_Dz_E+4y_E^2z_Dz_E+4x_Dx_Ez_E^2+4y_Dy_Ez_E^2+4z_Dz_E^3)+(R^4-2R^2x_E^2-2R^2y_E^2+2R^2z_E^2+r^4-2r^2R^2-2r^2x_E^2-2r^2y_E^2-2r^2z_E^2+x_E^4+y_E^4+z_E^4+2x_E^2y_E^2+2x_E^2z_E^2+2y_E^2z_E^2)=0$

Finding the normal to this torus requires finding the intersection point on the surface, $(x, y, z) = ((R + r\cos\phi)\cos\theta), (R + r\cos\phi)\sin\theta, r\sin\phi)$, and the nearest point on the ring through the middle of the torus, $(R\cos\theta, R\sin\theta, 0)$. Subtracting one from the other gives the normal as $\mathbf{N} = (r\cos\phi\cos\theta, r\cos\phi\sin\theta, r\sin\phi)$.

1.3.5 Plane

A plane can be defined by a normal vector, **N** and a point on the plane, **Q**. A point, **P**, is on the plane if:

$$\mathbf{N} \cdot (\mathbf{P} - \mathbf{Q}) = 0 \tag{57}$$

To find the ray/plane intersection substitute Equation 22 in Equation 57:

$$\mathbf{N} \cdot (\mathbf{E} + t\mathbf{D} - \mathbf{Q}) = 0 \tag{58}$$

$$\Rightarrow \quad t = \frac{\mathbf{N} \cdot (\mathbf{Q} - \mathbf{E})}{\mathbf{N} \cdot \mathbf{D}} \tag{59}$$

If $t < 0$ then the plane is behind the eye point and there is no intersection. If $t \geq 0$ then the intersection point is $\mathbf{E} + t\mathbf{D}$. If $\mathbf{N} \cdot \mathbf{D} = 0$ then the ray is parallel to the plane, and there is no intersection point. The normal vector is obviously just **N**.

1.3.6 Polygon

A polygon can be defined by an ordered set of vertices: $(\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}, \ldots)$. To find the intersection point between a polygon and a ray, we first find the intersection point between the polygon's plane and the ray, and then ascertain whether this intersection point lies inside the polygon or not.

The normal of the polygon's plane can be found by the simple cross product: $\mathbf{N} = (\mathbf{P_3} - \mathbf{P_2}) \times (\mathbf{P_1} - \mathbf{P_2})$. A point on the plane's surface is even easier to find: $\mathbf{Q} = \mathbf{P_1}$. The intersection calculation then proceeds as above.

If an intersection point between the ray and the plane is found then we can check whether or not the point lies inside the polygon in the following manner. First we project the intersection point and the polygon to two dimensions by simply throwing away one coordinate. Obviously we will throw away the coordinate in which the polygon has the smallest extent. We then test to see if the intersection point lies inside the two dimensional polygon using the odd/even test.

The odd/even test checks to see whether or not an arbitrary point lies inside an arbitrary polygon in two dimensions. This is done by drawing an arbitrary ray from the point to infinity. If the ray crosses an even number of polygon edges then the point lies outside the polygon. Contrariwise, if the ray crosses an odd number of polygon edges then the point lies inside the polygon. The easiest implementation of this simply runs the arbitrary ray parallel to one of the axes. A full discussion of the implementation details of this, and other point-in-polygon algorithms, can be found in *Graphics Gems IV* pp. 24–46.

1.3.7 Disc

The disc is similar to the polygon. Both are planar objects. A disc can be defined by its centre, **Q**, its radius, r, and a normal vector, **N**. Finding the intersection point, **P**, of the ray with the disc's plane proceeds as for a plane/ray or polygon/ray intersection. Discovering whether $\mathbf{P}$ lies inside the disc requires you to simply check that:

$$(\mathbf{P} - \mathbf{Q}) \cdot (\mathbf{P} - \mathbf{Q}) \leq r^2 \tag{60}$$

1.4 Intersections with arbitrarily positioned primitives

Of the above primitives, only the plane and polygon are arbitrarily defined[3]. The sphere, cylinder, cone, and torus are all defined as being centred at the origin, and all have other restrictions on their definition[4]. In order to ray trace one of these primitives in an arbitrary location we have two alternatives: (1) find general intersection algorithms between a ray and the arbitrarily located versions of the primitives; or (2) use geometric transforms to scale, rotate, and translate these primitives into the desired locations. This second option will be followed here.

1.4.1 Transforming the primitives

The basic idea is simple. We specify a scaling, a rotation, and a translation which, between them, transform the primitive from its standard position to the desired location. You will remember that this was covered in the IB *Computer Graphics & Image Processing* course. To perform the intersection we take the inverse transform of the ray, intersect this with the primitive in its standard position, and then transform the resulting intersection point to its correct location.

Remember that there is a difference between points and vectors. When transforming objects, vectors are scaled and rotated but *not* translated. Think of it this way: if you translate two points, the vector between them stays exactly the same. But if you scale or rotate the two points, the vector between them scales or rotates accordingly.

All this is by way of explaining how we transform a ray in order to intersect the appropriate ray with the primitive in its standard position. Let us assume that the primitive object in its standard position, $\hat{\mathbf{B}}$, undergoes transformation[5] $\mathbf{TRS}$ to get into the desired position $\mathbf{B} = \mathbf{TRS}\hat{\mathbf{B}}$. To intersect ray, $\mathbf{E} + t\mathbf{D}$, with $\mathbf{B}$ we transform the point, $\mathbf{E}$, and the vector $\mathbf{D}$ as follows:

$$\hat{\mathbf{E}} = \mathbf{S}^{-1}\mathbf{R}^{-1}\mathbf{T}^{-1}\mathbf{E} \tag{61}$$

$$\hat{\mathbf{D}} = \mathbf{S}^{-1}\mathbf{R}^{-1}\mathbf{D} \tag{62}$$

The point is translated, rotated and scaled but the vector is only rotated and scaled.

Now intersect ray, $\hat{\mathbf{E}} + t\hat{\mathbf{D}}$, with the object in its standard position, $\hat{\mathbf{B}}$, as described in the previous sections. This gives the value of t and consequently allows you to directly calculate $\mathbf{P} = \mathbf{E} + t\mathbf{D}$.

In addition to scaling, rotating and translating the standard primitives, this mechanism allows us to stretch and squash them by scaling them by different factors in the different dimensions. For example, an ellipsoid is simply a stretched sphere. However,

[3]The disc is also arbitrarily defined if you want a circular disc, but it could be transformed to provide an ellipse.

[4]The sphere has unit radius; the cylinder has unit radius and is aligned with the z-axis; the cone is similarly aligned and has unit slope; the torus is aligned with the z-axis.

[5]Where $\mathbf{T}$ is a translation, $\mathbf{R}$ a rotation, and $\mathbf{S}$ a scaling.

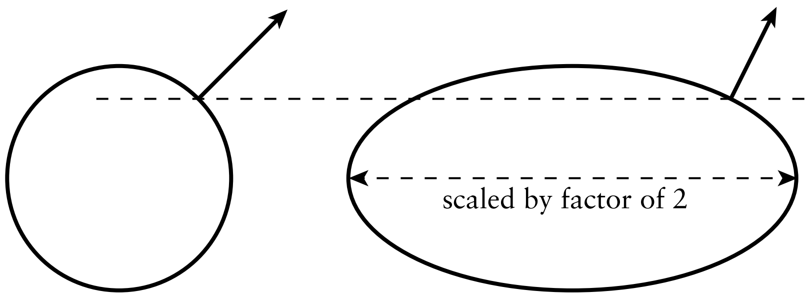


Figure 1: At left is a circle and a normal vector to the circle at a particular point. If we scale the circle by a factor of two horizontally, the normal vector scales by a factor of a half in the same dimension.

such anisotropic scaling causes interesting problems with the normal vectors, as discussed in the following section.

1.4.2 Transforming normal vectors

In ray tracing we need to know not just the intersection point but also the normal vector to the surface at the intersection point. This normal vector is used in the illumination calculations. However, if we scale an object anisotropically (different amounts in each coordinate), the normal vector scales as the *inverse* of the object scaling, although it rotates in the *same* way as the object. Figure 1 graphically illustrates this phenomenon.

This can be mathematically explained by remembering that the normal vector is related to the derivative of the surface. The normal vector is perpendicular to the surface, which means that it is perpendicular to any derivative vector. As an example, consider the ellipsoid created by scaling the unit sphere by the matrix:

$$\mathbf{S} = \begin{bmatrix} l & 0 & 0 & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{63}$$

The intersection point between the unit sphere and the inverse transformed ray will be at some point $\hat{\mathbf{P}} = (\hat{x}, \hat{y}, \hat{z})$. This equates to the spherical polar coordinate $(1, \theta, \phi)$ where[6] $(\hat{x}, \hat{y}, \hat{z}) = (\cos\phi\cos\theta, \cos\phi\sin\theta, \sin\phi)$. By virtue of the fact that the normal to every point on the sphere passes through the centre of the sphere, it is easy to see that the normal vector, $\hat{\mathbf{N}}$, is also $\hat{\mathbf{N}} = (\cos\phi\cos\theta, \cos\phi\sin\theta, \sin\phi)$.

Transforming $\hat{\mathbf{P}}$ to the true intersection point $\mathbf{P}$ is simply a matter of applying $\mathbf{P} = \mathbf{S}\hat{\mathbf{P}}$. Thus the intersection point of the ray with the ellipsoid is at rectangular coordinate $(x, y, z) = (l\cos\phi\cos\theta, m\cos\phi\sin\theta, n\sin\phi)$.

To find the normal vector to the ellipsoid at this intersection point, we need to find two non-parallel derivative vectors to the surface at the intersection point, and take

[6]If you are wondering where r has disappeared to, remember that, in a unit sphere, $r = 1$.

their cross product to give the normal vector, **N**. Two derivative vectors are:

$$\frac{\partial(\mathbf{P})}{\partial\phi} = \frac{\partial(x,y,z)}{\partial\phi} = (-l\sin\phi\cos\theta, -m\sin\phi\sin\theta, n\cos\phi) \tag{64}$$

$$\frac{\partial(\mathbf{P})}{\partial\theta} = \frac{\partial(x,y,z)}{\partial\theta} = (-l\cos\phi\sin\theta, m\cos\phi\cos\theta, 0) \tag{65}$$

This leads to the normal vector being[7]:

$$\mathbf{N} = \frac{\partial(\mathbf{P})}{\partial\phi} \times \frac{\partial(\mathbf{P})}{\partial\theta} \tag{66}$$

$$= (\frac{1}{l}\cos\phi\cos\theta, \frac{1}{m}\cos\phi\sin\theta, \frac{1}{n}\sin\phi) \tag{67}$$

Thus, we conclude that while:

$$\mathbf{P} = \mathbf{S}\hat{\mathbf{P}} \tag{68}$$

$$\mathbf{N} = \mathbf{S}^{-1}\hat{\mathbf{N}} \tag{69}$$

This leaves open the question of what to do about the rotation and translation components of the transformation. It is intuitively obvious that rotating an object causes the normal vector to rotate by the same amount and, because normal vectors are vectors rather than points, they should not be translated. So the final analysis is that:

$$\mathbf{P} = \mathbf{TRS}\hat{\mathbf{P}} \tag{70}$$

$$\mathbf{N} = \mathbf{RS}^{-1}\hat{\mathbf{N}} \tag{71}$$

This analysis has been applied to the unit sphere, but the same result holds for any object.

1.5 Converting the primitives to polygons

We may be in a situation where we define objects in terms of the various primitives, but where we wish to draw the objects using polygon scan conversion. In this case we need to convert primitives into polygons. Some polygon scan conversion algorithms can only deal with triangles; in these cases we may need to do a little bit of extra work to ensure that all of the generated polygons are triangles.

Converting the curved primitives (sphere, cylinder, cone, torus, disc) involves approximating a curved profile by a series of straight line segments. The simplest example is the disc. This can be approximated by a regular n-gon, where n is chosen to give an adequate approximation to the disc. "Adequate" in this case will depend on the desired rendering resolution, the desired speed of rendering, and the desired quality of the final image. If necessary, this n-gon can be converted to triangles in one of a number of ways, as shown in Figure 2.

A cylinder or cone can be converted to polygons by polygonising the discs at either end into n-gons, and then connecting corresponding vertices on the two n-gons. In the special case of a cone with a point, one of the n-gons obviously degenerates to that point.

[7] If you try this at home you will find that you will need to divide through by the factor $-1/(lmn\cos\phi)$.

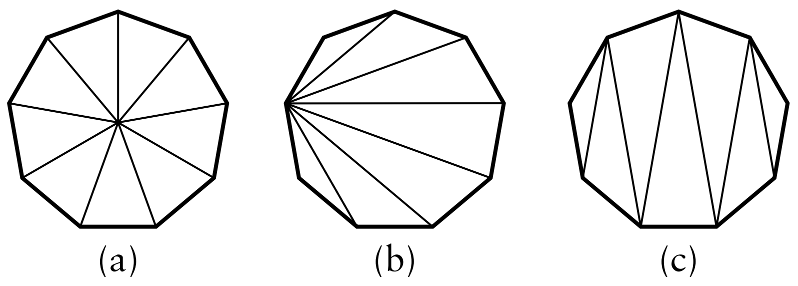


Figure 2: Three ways to split a polygon into triangles: (a) define a central vertex and connect every edge to this vertex to make n isosceles triangles; (b) select one vertex on the n-gon, and make a triangle fan emanating from this vertex; or (c) start at one edge of the n-gon and make a triangle strip set which proceeds from this edge of the polygon to the opposite edge.

Spheres and tori can be most easily converted to polygons by considering their parameterisation in terms of θ and ϕ. For a sphere these are Equations 72–74 (below); for a torus they are Equations 53–55.

$$x = r\cos\phi\cos\theta \tag{72}$$
$$y = r\cos\phi\sin\theta \tag{73}$$
$$z = r\sin\phi \tag{74}$$

By selecting appropriate steps in the two parameters we can generate a set of quadrilaterals which approximates the curved primitive.

It should be noted that spheres can be polygonised with more uniform polygons by starting with one of the five Platonic solids, and subdividing its faces accordingly. The details of this are left as an exercise to the reader.

2 Bezier curves

A Bezier curve is a weighted sum of $n+1$ control points, $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$, where the weights are the Bernstein polynomials:

$$\mathbf{P}(t) = \sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) (1-t)^{n-i} t^i \mathbf{P}_i, 0 \le t \le 1 \tag{75}$$

The Bezier curve of order $n+1$ (degree n) has $n+1$ control points. Below are the first three orders of Bezier curve definitions.

$$\text{linear} \quad \mathbf{P}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \tag{76}$$
$$\text{quadratic} \quad \mathbf{P}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \tag{77}$$
$$\text{cubic} \quad \mathbf{P}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \tag{78}$$

2.1 Ways of thinking about Bezier curves

There are several useful ways in which you can think about Bezier curves. Here are the ones that I use.

Linear interpolation. Equation 76 is obviously a linear interpolation between two points. Equation 77 can be rewritten as a linear interpolation between linear interpolations between points:

$$\mathbf{P}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2] \qquad (79)$$

Equation 78 can be rewritten as a linear interpolation between linear interpolations between linear interpolations between points. This is left as an exercise for the reader.

Weighted average. A Bezier curve can be seen as a weighted average of all of its control points. Because all of the weights are positive, and because the weights sum to one, the Bezier curve is guaranteed to lie within the convex hull of its control points.

Refinement of the control polygon. A Bezier curve can be seen as some sort of refinement of the polygon made by connecting its control points in order. The Bezier curve starts and ends at the two end points and its shape is determined by the relative positions of the $n-1$ other control points, although it will generally not pass through any of these other control points. The tangent vectors at the start and end of the curve pass through the end point and the immediately adjacent point.

Rogers and Adams list the properties of the Bezier curve on page 291.

2.2 Continuity

You should note that each Bezier curve is independent of any other Bezier curve. If we wish two Bezier curves to join with any type of continuity, then we must explicitly position the control points of the second curve so that they bear the appropriate relationship with the control points in the first curve.

Any Bezier curve is infinitely differentiable within itself, and is therefore continuous to any degree (C^n-continuous, $\forall n$). We therefore only need concern ourselves with continuity across the joins between curves. Assume that we have two Bezier curves of the same order: $\mathbf{P}(t)$, defined by $(\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n)$, and $\mathbf{Q}(t)$, defined by $(\mathbf{Q}_0, \mathbf{Q}_1, \ldots, \mathbf{Q}_n)$. C^0-continuity (continuity of position) can be achieved by setting $\mathbf{P}(1) = \mathbf{Q}(0)$. This gives a formula for $\mathbf{Q}_0$ in terms of the $\mathbf{P}_i$s:

$$\mathbf{Q}_0 = \mathbf{P}_n. \qquad (80)$$

Similarly for C^1-continuity, we need C^0-continuity and $\mathbf{P}'(1) = \mathbf{Q}'(0)$, giving:

$$\mathbf{Q}_1 - \mathbf{Q}_0 = \mathbf{P}_n - \mathbf{P}_{n-1} \qquad (81)$$

Combining Equations 81 and 80 gives a formula for $\mathbf{Q}_1$ in terms of the $\mathbf{P}_i$s:

$$\begin{aligned} \mathbf{Q}_1 &= 2\mathbf{P}_n - \mathbf{P}_{n-1} && (82) \\ &= \mathbf{P}_n + (\mathbf{P}_n - \mathbf{P}_{n-1}) && (83) \end{aligned}$$

Continuing in this vein, we find that the requirements for C^2-continuity (i.e. C^1-continuity and $\mathbf{P}''(1) = \mathbf{Q}''(0)$) give:

$$\mathbf{Q}_2 - 2\mathbf{Q}_1 + \mathbf{Q}_0 = \mathbf{P}_n - 2\mathbf{P}_{n-1} + \mathbf{P}_{n-2} \tag{84}$$

Combining Equations 84, 81, and 80 gives a formula for $\mathbf{Q}_2$ in terms of the $\mathbf{P}_i$s:

$$\begin{aligned} \mathbf{Q}_2 &= 4\mathbf{P}_n - 4\mathbf{P}_{n-1} + \mathbf{P}_{n-2} && (85) \\ &= \mathbf{P}_{n-2} + 4(\mathbf{P}_n - \mathbf{P}_{n-1}) && (86) \end{aligned}$$

2.3 Bezier surfaces

We learnt in the IB course that the simplest way to construct a Bezier surface is as the tensor product of Bezier curves. A tensor product Bezier surface of order $n+1$ is defined by $(n+1)^2$ control points. It is called a Bezier patch.

$$\mathbf{P}(s,t) = \sum_{i=0}^{n} \binom{n}{i} (1-s)^{n-i} s^i \sum_{j=0}^{n} \binom{n}{j} (1-t)^{n-j} t^j \mathbf{P}_{i,j} \tag{87}$$

You can think about this as moving the control points of one Bezier curve along a set of Bezier curves to sweep out a surface. Continuity across a boundary between two Bezier patches is only guaranteed if each of the Bezier curves across the join obey the curve continuity conditions. Again, this was covered in the IB course.

3 B-splines

B-splines are a more general type of curve than Bezier curves. In a B-spline each control point is associated with a *basis function*.

$$\mathbf{P}(t) = \sum_{i=1}^{n+1} N_{i,k}(t)\mathbf{P}_i, t_{\min} \le t < t_{\max} \tag{88}$$

There are $n+1$ control points, $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_{n+1}$. The $N_{i,k}$ basis functions are of order k (degree $k-1$). k must be at least 2 (linear), and can be no more than $n+1$ (the number of control points). The important point here is that the order of the curve (2 [linear], 3 [quadratic], 4 [cubic],...) is therefore not dependent on the number of control points (which it is for Bezier curves, where k must always equal $n+1$).

Equation 88 defines a piecewise continuous function. A *knot vector*, $(t_1, t_2, \ldots, t_{k+(n+1)})$, must be specified. This determines the values of t at which the pieces of curve join, like knots joining bits of string. It is necessary that:

$$t_i \le t_{i+1}, \forall i \tag{89}$$

The $N_{i,k}$ depend *only* on the value of k and the values, t_i, in the knot vector. $N_{i,k}$ is defined recursively as:

$$\begin{aligned} N_{i,1}(t) &= \begin{cases} 1, & t_i \le t < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \\ N_{i,k}(t) &= \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) && (90) \end{aligned}$$

This is essentially a modified version of the idea of taking linear interpolations of linear interpolations of linear interpolations...

At this point it would be instructive for you to work out $N_{1,1}$, $N_{2,1}$, $N_{3,1}$, $N_{1,2}$, $N_{2,2}$, $N_{1,3}$ for the knot vector $(0, 2, 3, 6)$. It helps if you draw the graphs for these functions.

There are several things that you should note about these equations. Each $N_{i,k}(t)$ depends only on the $k + 1$ knot values from t_i to t_{i+k}. $N_{i,k}(t) = 0$ for $t < t_i$ or $t \geq t_{i+k}$ so $\mathbf{P}_i$ only influences the curve for $t_i \leq t < t_{i+k}$. Formally, $\mathbf{P}(t)$ is a polynomial of order k (degree $k - 1$) on each interval $t_i \leq t < t_{i+1}$. Across the knots $\mathbf{P}(t)$ is C^{k-2}-continuous. $\mathbf{P}(t)$ is, of course, continuous in all its derivatives between the knots. Remember from Equation 16 that a weighted sum of points only makes sense if the weights sum to one. $\mathbf{P}(t)$ is therefore validly defined only where

$$\sum_{i=1}^{n+1} N_{i,k}(t) = 1. \qquad (91)$$

This is the range $t_{\min} \leq t < t_{\max}$ where $t_{\min} = t_k$ and $t_{\max} = t_{n+2}$. Even more properties of B-splines are described in Rogers and Adams pp. 306–7.

3.1 The knot vector

The above explanation shows that the knot vector is very important. The knot vector can, by its definition, be any sequence of numbers provided that each one is greater than or equal to the preceding one. Some types of knot vector are more useful than others. Knot vectors are generally placed into one of three categories: uniform, open uniform, and non-uniform.

Uniform. These are knot vectors for which

$$t_{i+1} - t_i = \text{constant}, \forall i \qquad (92)$$

For example:

$$\begin{array}{l} [1, 2, 3, 4, 5, 6, 7, 8] \\ [0, 1, 2, 3, 4, 5] \\ [0, 0.25, 0.5, 0.75, 1.0] \\ [-2.5, -1.4, -0.3, 0.8, 1.9, 3.0] \end{array}$$

All of the basis functions are just shifted versions of one another and so the implementation is very easy.

Open Uniform. These are uniform knot vectors which have k equal knot values at each end:

$$\begin{array}{rl} t_i = t_1, & i \leq k \\ t_{i+1} - t_i = \text{constant}, & k \leq i < n+2 \\ t_i = t_{k+(n+1)}, & i \geq n+2 \end{array} \qquad (93)$$

For example:

$$\begin{array}{ll} [0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4] & (k = 4) \\ [1, 1, 1, 2, 3, 4, 5, 6, 6, 6] & (k = 3) \\ [0.1, 0.1, 0.1, 0.1, 0.1, 0.3, 0.5, 0.7, 0.7, 0.7, 0.7, 0.7] & (k = 5) \end{array}$$

This is essentially just a simple modification to the uniform case which allows the curve to go through its two end points.

Non-uniform. This is the general case, the only constraint being the standard $t_i \leq t_{i+1}, \forall i$ (Equations 89). For example:

$$
\begin{array}{l}
[1, 3, 7, 22, 23, 23, 49, 50, 50] \\
[1, 1, 1, 2, 2, 3, 4, 5, 6, 6, 6, 7, 7, 7] \\
[0.2, 0.7, 0.7, 0.7, 1.2, 1.2, 2.9, 3.6]
\end{array}
$$

The shapes of the $N_{i,k}$ basis functions are determined entirely by the *relative* spacing between the knots. Scaling ($t'_i = \alpha t_i, \forall i$) or translating ($t'_i = t_i + \Delta t, \forall i$) the knot vector has no effect on the shapes of the $N_{i,k}$.

The above gives a description of the various types of knot vector but it doesn't really give you any insight into how the knot vector determines the shape of the curve. The following subsections look at the different types of knot vector in more detail. However, the best way to get to feel for these is to derive and draw the basis functions yourself, and to play with them on a computer using some sort of tutorial tool (see the Study Guide for details of a Bezier and a NURBS tutorial tool available on the web).

3.1.1 Uniform knot vector

For simplicity, let $t_i = i$ (this is allowable given that the scaling or translating the knot vector has no effect on the shapes of the $N_{i,k}$). The knot vector thus becomes $[1, 2, 3, \ldots, k + (n + 1)]$ and Equation 90 simplifies to:

$$
\begin{aligned}
N_{i,1}(t) &= \begin{cases} 1, & i \leq t < i + 1 \\ 0, & \text{otherwise} \end{cases} \\
N_{i,k}(t) &= \frac{t - i}{k - 1} N_{i,k-1}(t) + \frac{i + k - t}{k - 1} N_{i+1,k-1}(t)
\end{aligned}
\tag{94}
$$

You should be easily able to graph the first few of these for yourself. The principle thing to note about the uniform basis functions is that, for a given order k, the basis functions are all simply shifted versions of one another. See Rogers and Adams Figure 5-36.

3.1.2 Things you can change about a uniform B-spline

With a uniform B-spline, you obviously cannot change the basis functions (they are fixed because all the knots are equispaced). However you can alter the shape of the curve by modifying a number of things:

Moving control points. Moving the control points obviously changes the shape of the curve.

Multiple control points. Sticking two adjacent control points on top of one another causes the curve to pass closer to that point. Stick enough adjacent control points on top of one another and you can make the curve pass through that point (Rogers and Adams, Figure 5-45).

Order. Increasing the order k increases the continuity of the curve at the knots, increases the smoothness of the curve, and tends to move the curve farther from its defining polygon. (Rogers and Adams, Figure 5-44).

Joining the ends. You can join the ends of the curve to make a closed loop. Say you have M points, $\mathbf{P}_1, \ldots, \mathbf{P}_M$. You want a closed B-spline defined by these points. For a given order, k, you will need $M + (k-1)$ control points (repeating the first $k-1$ points): $\mathbf{P}_1, \ldots, \mathbf{P}_M, \mathbf{P}_1, \ldots, \mathbf{P}_{k-1}$. Your knot vector will thus have $M + 2k - 1$ uniformly spaced knots.

3.1.3 Open uniform knot vector

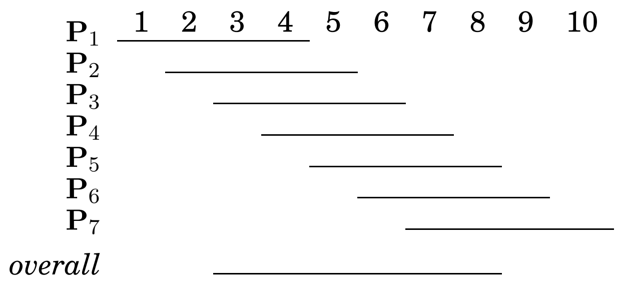
The previous section intimated that uniform B-splines can be used to describe closed curves: all you have to do is join the ends as described above. If you do not want a closed curve, and you use a uniform knot vector, you find that you need to specify control points at each end of the curve which the curve doesn't go near (e.g. Rogers and Adams, Figure 5-44, the order 4 curve).

If you wish your B-spline to start and end at your first and last control points then you need an open uniform knot vector (e.g. Rogers and Adams, Figure 5-41). The only difference between this and the uniform knot vector being that the open uniform version has k equal knots at each end.

An order k open uniform B-spline with $n+1 = k$ points is the Bezier curve of order k. It would be a useful exercise for you to prove this for $k = 3$. For ease of calculation take the knot vector to be $[0, 0, 0, 1, 1, 1]$.

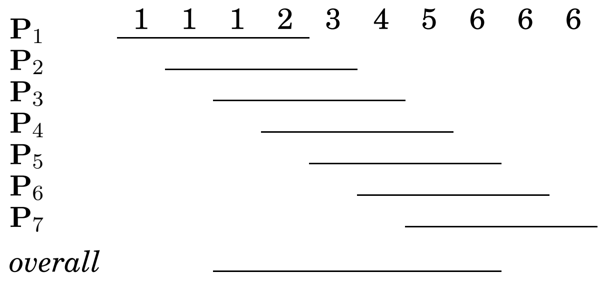
3.1.4 The difference between uniform and open uniform

It may help, at this stage, to compare a particular uniform and an equivalent open uniform knot vector. This is a uniform knot vector for $n + 1 = 7$, $k = 3$:



The lines show the range of t over which each $\mathbf{P}_i$ is non-zero. The B-spline itself (the *overall* line in the diagram) is defined over the range $t_3 \leq t < t_8$, i.e. over the range $3 \leq t < 8$.

By comparison an open uniform knot vector for $n + 1 = 7$, $k = 3$ is:



The B-spline itself is defined over the range $t_3 \leq t < t_8$, i.e. over the range $1 \leq t < 6$. By the definition of a open uniform knot vector $t_3 = t_1$ and $t_8 = t_{10}$ and so an open uniform B-spline is defined over the *full* range of t from t_1 to $t_k + n + 1$.

3.1.5 Non-uniform knot vector

Any B-spline whose knot vector is neither uniform nor open uniform is non-uniform. Non-uniform knot vectors allow any spacing of the knots, including multiple knots (adjacent knots with the same value). We need to know how this non-uniform spacing affects the basis functions in order to understand where non-uniform knot vectors could be useful. It transpires that there are only three cases of any interest: (1) multiple knots (adjacent knots equal); (2) adjacent knots more closely spaced than the next knot in the vector; and (3) adjacent knots less closely spaced than the next knot in the vector. Obviously, case (3) is simply case (2) turned the other way round.

Multiple knots. A multiple knot reduces the degree of continuity at that knot value. Across a normal knot the continuity is C^{k-2}. Each extra knot with the same value reduces continuity at that value by one. This is the only way to reduce the continuity of the curve at the knot values. If there are $k - 1$ (or more) equal knots then you get a discontinuity in the curve.

Close knots. As two knots' values get closer together, relative to the spacing of the other knots, the curve moves closer to the related control point.

Distant knots. As two knots' values get further apart, relative to the spacing of the other knots, the curve moves further away from the related control point.

3.1.6 Use of non-uniform knot vectors

Standard procedure is to use uniform or open uniform B-splines unless there is a very good reason not to do so. Moving two knots closer together tends to move the curve only slightly and so there is usually little point in doing it. This leads to the conclusion that the main use of non-uniform B-splines is to allow for multiple knots, which adjust the continuity of the curve at the knot values.

However, non-uniform B-splines are the general form of the B-spline because they incorporate open uniform and uniform B-splines as special cases. Thus we will talk about *non-uniform B-splines* when we mean the general case, incorporating both uniform and open uniform.

3.1.7 What can you do to control the shape of a B-spline?

- Move the control points.
- Add or remove control points.
- Use multiple control points.
- Change the order, k.
- Change the type of knot vector.

- Change the relative spacing of the knots.
- Use multiple knot values in the knot vector.

3.1.8 What should the defaults be?

If there are no pressing reasons for doing otherwise, your B-spline should be defined as follows:

- $k = 4$ (cubic);
- no multiple control points;
- uniform (for a closed curve) or open uniform (for an open curve) knot vector.

3.2 B-spline patches

We generalise from B-spline curves to B-spline surfaces in the same way as we did for Bezier patches. Take a tensor product of two versions of Equation 88.

$$\mathbf{P}(s,t) = \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} \mathbf{P}_{i,j} N_{i,k}(s) N_{j,l}(t), s_{\min} \le s < s_{\max}, t_{\min} \le t < t_{\max} \tag{95}$$

where it is usual for the patch to have the same order (i.e. $k = l$) in both directions. Patches are thus defined by a quadrilateral grid of control points of size $(m+1) \times (n+1)$.

4 Rational B-splines

Rational B-splines have all of the properties of non-rational B-splines plus the following two useful features:

- They produce the correct results under projective transformations (while non-rational B-splines only produce the correct results under affine transformations).
- They can be used to represent lines, conics, non-rational B-splines; and, when generalised to patches, can represent planes, quadrics, and tori.

In this case *rational* means “one polynomial divided by another” (see Equation 96). The antonym of *rational* is *non-rational* (i.e. a non-rational B-spline is just a polynomial (i.e. Equation 88). Non-rational B-splines are a special case of rational B-splines, just as uniform B-splines are a special case of non-uniform B-splines. Thus, *non-uniform rational B-splines* encompass almost every other possible 3D shape definition. *Non-uniform rational B-spline* is a bit of a mouthful and so it is generally abbreviated to *NURBS*.

We have already learnt all about the the *B-spline* bit of *NURBS* and about the *non-uniform* bit. So now all we need to know is the meaning of the *rational* bit and we will fully(?) understand NURBS.

Rational B-splines are defined simply by applying the B-spline equation (Equation 88) to homogeneous coordinates, rather than normal 3D coordinates. We discussed homogeneous coordinates in the IB course. You will remember that these are 4D coordinates where the transformation from 4D to 3D is:

$$(x', y', z', w) \to \left(\frac{x'}{w}, \frac{y'}{w}, \frac{z'}{w}\right) \tag{96}$$

Last year we said that the inverse transform was:

$$(x, y, z) \to (x, y, z, 1) \tag{97}$$

This year we are going to be more cunning and say that:

$$(x, y, z) \to (xh, yh, zh, h) \tag{98}$$

Thus our 3D control point, $\mathbf{P}_i = (x_i, y_i, z_i)$, becomes the homogeneous control point, $\mathbf{C}_i = (x_i h_i, y_i h_i, z_i h_i, h_i)$.

A NURBS curve is thus defined as:

$$\mathbf{P}_H(t) = \sum_{i=1}^{n+1} N_{i,k}(t)\mathbf{C}_i, t_{\min} \leq t < t_{\max} \tag{99}$$

Compare Equation 99 with Equation 88 to see just how easy this is!

We now want to see what a NURBS curve looks like in normal 3D coordinates, so we need to apply Equation 96 to Equation 99. In order to better explain what is going on, we first write Equation 99 in terms of its individual components. Equation 99 is equivalent to:

$$\begin{aligned} x'(t) &= \sum_{i=1}^{n+1} x_i h_i N_{i,k}(t) & (100)\\ y'(t) &= \sum_{i=1}^{n+1} y_i h_i N_{i,k}(t) & (101)\\ z'(t) &= \sum_{i=1}^{n+1} z_i h_i N_{i,k}(t) & (102)\\ h(t) &= \sum_{i=1}^{n+1} h_i N_{i,k}(t) & (103) \end{aligned}$$

Equation 96 tells us that, in 3D:

$$\begin{aligned} x(t) &= x'(t)/h(t) & (104)\\ y(t) &= y'(t)/h(t) & (105)\\ z(t) &= z'(t)/h(t) & (106) \end{aligned}$$

Thus the 4D to 3D conversion gives us the curve in 3D:

$$\mathbf{P}(t) = \frac{\sum_{i=1}^{n+1} N_{i,k}(t)\mathbf{P}_i h_i}{\sum_{i=1}^{n+1} N_{i,k}(t) h_i}, t_{\min} \leq t < t_{\max} \tag{107}$$

This looks a lot more fierce than Equation 99, but is simply the same thing written a different way.

So now, we need to define an additional parameter, h_i, for each control point, $\mathbf{P}_i$. The default is to set $h_i = 1, \forall i$. This results in the denominator of Equation 107 becoming one, and the NURBS equation (Equation 107) therefore reducing to the non-rational B-spline equation (Equation 88).

Increasing h_i pulls the curve closer to point $\mathbf{P}_i$. Decreasing h_i pushes the curve farther from point $\mathbf{P}_i$. Setting $h_i = 0$ means that $\mathbf{P}_i$ has no effect on the curve at all. See Rogers and Adams Figure 5-58 for an example, and play with an on-line NURBS tutorials such as the one mentioned in the Study Guide.

4.1 An example: a circle defined by NURBS

A non-rational B-spline or a Bezier curve cannot exactly represent a circle. An interesting exercise is to place a cubic Bezier curve's end points at $(0,1)$ and $(1,0)$, with the other control points at $(\alpha,1)$ and $(1,\alpha)$. Now see how close this "quarter circle" comes to the real quarter circle defined by $x^2+y^2=1$, i.e. what is the value of α for which the Bezier curve most closely matches the quarter circle.

NURBS *can* be used to represent circles, and all of the other conics. NURBS surfaces can be used to represent quadric surfaces. As an example, let us consider one way in which NURBS can be used to describe a true circle. Rogers and Adams cover this on pages 371–375.

Construct eight control points in a square. Let $\mathbf{P}_1$, $\mathbf{P}_3$, $\mathbf{P}_5$, and $\mathbf{P}_7$ be the vertices of the square. Let $\mathbf{P}_0$, $\mathbf{P}_2$, $\mathbf{P}_4$, and $\mathbf{P}_6$ be the midpoints of the respective sides, so that the vertices are numbered sequentially as you proceed around the square. Finally, you need a ninth point to join up the curve, so let $\mathbf{P}_8 = \mathbf{P}_0$.

Use a quadratic B-spline basis function with the knot vector
$[0,0,0,1,1,2,2,3,3,4,4,4]$. This means that the curve will pass through $\mathbf{P}_0$, $\mathbf{P}_2$, $\mathbf{P}_4$, $\mathbf{P}_6$ and $\mathbf{P}_8$, and allows us to essentially treat each quarter of the circle independently. That is, we can take just examine $\mathbf{P}_0$, $\mathbf{P}_1$, and $\mathbf{P}_2$, along with the knot vector $[0,0,0,1,1,1]$. If this makes a quarter circle then the other three quarters will also be correct.

We finally need to specify the homogeneous co-ordinates. As a circle is symmetrical it should be obvious that that $h_1 = h_3 = h_5 = h_7 = \alpha$ and $h_0 = h_2 = h_4 = h_6 = h_8 = \beta$. As we would like the curve to pass through the even numbered points we know that $\beta = 1$. All we therefore need to determine is α, the value of the odd numbered homogeneous co-ordinates.

If $\alpha = 1$ then the NURBS curve will bulge out more than a circle. If $\alpha = 0$, it will bow in. This gives us limits on the value of α. To find the exact value we take one quarter of the NURBS curve definition:

$$\mathbf{P}(t) = \frac{(1-t)^2\mathbf{P}_0 + 2\alpha t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2}{(1-t)^2 + 2\alpha t(1-t) + t^2}, 0 \le t < 1 \qquad (108)$$

Assume now that $\mathbf{P}_0 = (0,1)$, $\mathbf{P}_1 = (1,1)$, and $\mathbf{P}_2 = (1,0)$. Insert Equation 108 into the equation for the unit circle ($x(t)^2 + y(t)^2 = 1$). The resulting equation is:

$$\frac{((1-t)^2 + 2\alpha t(1-t))^2 + (2\alpha t(1-t) + t^2)^2}{((1-t)^2 + 2\alpha t(1-t) + t^2)^2} = 1, 0 \le t < 1 \qquad (109)$$

Now solve this for α. Equation 109 is essentially:

$$\frac{a_N t^4 + b_N t^3 + c_N t^2 + d_N t + e_N}{a_D t^4 + b_D t^3 + c_D t^2 + d_D t + e_D} = 1, 0 \leq t < 1 \tag{110}$$

From this we can conclude that we require $a_N = a_D$, $b_N = b_D$, $c_N = c_D$, $d_N = d_D$, and $e_N = e_D$. The first three all solve to give the result that $\alpha = 1/\sqrt{2}$, while the last two cancel out totally to give the tautology $0 = 0$. Thus $\alpha = 1/\sqrt{2}$. If you are in a hurry to solve this, and you can remember that each of the first three equations gives the same answer then you need simply extract the coefficients of t^4 from Equation 109 and solve. It is even easier if you remember the magic number, $1/\sqrt{2}$.

This derivation is not at all intuitive and similar cleverness is required to handle representations of other conics. The beauty of NURBS is that they allow us to do this sort of thing and unify all shapes into a single representation. The difficulty is that, in order to achieve this unification, we need to have this rather complicated but general mathematical mechanism.

4.2 Sweeping with NURBS

A NURBS surface is defined as the same two-dimensional extension to NURBS curves described in Equation 95, though obviously carried out in homogeneous co-ordinates. You can define sweeps using NURBS curves by using one NURBS curve as the sweep path, and another NURBS curve as the cross-section. You take the tensor product of the two curves. This will be described more fully in the lectures and is covered in Rogers & Adams, pages 445–456, 465–477.

5 An introduction to subdivision curves and surfaces

Subdivision schemes are increasingly being used as an alternative to NURBS. They combine mathematical elegance with an exceptionally simple implementation. For curves, given an arbitrary control polygon, we use the positions of the current vertices to determine the location of the new vertices in a new, refined, more detailed, control polygon. Generally, each old vertex gives rise to two new vertices. For example, you could place new vertices one-quarter and three-quarters of the way between each adjacent pair of old vertices. Connecting all the new vertices together, in the appropriate order, produces a more refined control polygon. Repeat this process several times and you produce a very good approximation to the uniform quadratic B-spline curve defined by the original set of vertices. In the limit, the refined control polygon becomes this uniform quadratic B-spline curve. The Doo-Sabin subdivision method is the extension of this idea to surfaces, where the refined control polygon has four times as many vertices as the source control polygon. Given the simplicity of the implementation and the fact that you can stop whenever you like, you can see how attractive this method is for computer graphics.

5.1 Mathematical details: curves

Take an arbitrary polygon defined by the sequence of control points:

$$\mathbf{P}^i = (\ldots, \mathbf{p}^i_{-1}, \mathbf{p}^i_0, \mathbf{p}^i_1, \mathbf{p}^i_2, \ldots)$$

Subdivision maps this sequence of control points to a new sequence, $\mathbf{P}^{i+1}$by applying subdivision rules. This process doubles[8] the number of points, and there is one rule for the odd numbered points and one for the even. For example, the subdivision rules on which the Doo-Sabin method is based are:

$$\mathbf{p}_{2j}^{i+1} = \frac{3}{4}\mathbf{p}_j^i + \frac{1}{4}\mathbf{p}_{j+1}^i \tag{111}$$

$$\mathbf{p}_{2j+1}^{i+1} = \frac{1}{4}\mathbf{p}_j^i + \frac{3}{4}\mathbf{p}_{j+1}^i \tag{112}$$

while the subdivision rules on which the Catmull-Clark method is based are:

$$\mathbf{p}_{2j}^{i+1} = \frac{1}{8}\mathbf{p}_{j-1}^i + \frac{6}{8}\mathbf{p}_j^i + \frac{1}{8}\mathbf{p}_{j+1}^i \tag{113}$$

$$\mathbf{p}_{2j+1}^{i+1} = \frac{4}{8}\mathbf{p}_j^i + \frac{4}{8}\mathbf{p}_{j+1}^i \tag{114}$$

As is the way with much mathematics, we can write it in a more compact, more general, but less obvious, form as:

$$\mathbf{p}_j^{i+1} = \sum_{k=-\infty}^{\infty} \alpha_{2k-j}\mathbf{p}_k^i \tag{115}$$

where the α_j are coefficients depending on the subdivision rules. Notes that the index $2k-j$ alternately selects the even indexed α_j and the odd indexed α_j. So, the two schemes given above, can be compactly described as:

$$\alpha = \frac{1}{4}(\ldots, 0, 0, 1, 3, 3, 1, 0, 0, \ldots) \tag{116}$$

and

$$\alpha = \frac{1}{8}(\ldots, 0, 0, 1, 4, 6, 4, 1, 0, 0, \ldots) \tag{117}$$

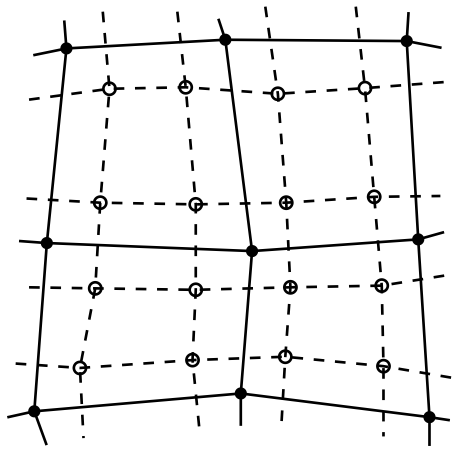
respectively. You will recognise the sequences in parentheses as being two rows from Pascal's triangle.

It would now be constructive for you to draw an arbitrary control polygon and perform a couple of subdivision steps using the first of the two subdivision schemes. Once you feel happy that you understand what is going on, you may like to try the second scheme. For those for whom these two tasks seem simple, you may like to consider what happens if you try to use the previous row from Pascal's triangle (1,2,1) and, for even more excitement, what happens if you try to use the next row (1,5,10,10,5,1). Both produce valid subdivision methods, but you will find that (1,2,1) has a minimal effect on the *shape* of the control polygon.

5.2 Mathematical details: surfaces

The above subdivision methods can be easily extended from a control polygon to a quadrilateral mesh. This is a mesh where every polygon is a quadrilateral and every vertex is connected to four other vertices.

[8]It doesn't quite double the number of points when the sequence is open and of finite length, but we will gloss over that at the moment.



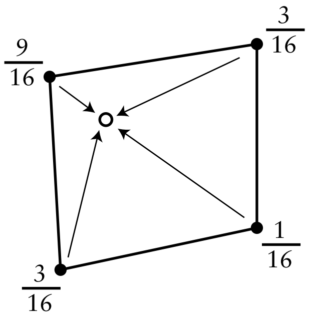


Figure 3: Doo-Sabin subdivision. On left a mesh (solid dots and solid lines) that has been refined (open dots and dashed lines). At right the weights used to generated one of the refined vertices.

The Doo-Sabin subdivision method introduces four new vertices in each quadrilateral, and connects up vertices accordingly. The new vertices are blended mixtures of the old vertices in the proportions $9 : 3 : 3 : 1$ (derived from the tensor product of the univariate case: $3 \times 3 : 3 \times 1 : 1 \times 3 : 1 \times 1$). This is illustrated in Figure 3.

Catmull-Clark subdivision is not much more difficult to understand. The only difference here is that is not all of the new vertices are created using the same weights. A vertex is introduced in the centre of each quadrilateral, in the centre of each edge, and near to each old vertex. Each of these three types of vertex has a different set of weights as illustrated in Figure 4.

This all works beautifully for quadrilateral meshes. Now, suppose we have a quadrilateral mesh that contains extraordinary vertices, in other words a mesh that consists of quadrilaterals but has occasional vertices with other than four immediate neighbours. The Doo-Sabin scheme will still worked quite happily, because every polygon in the mesh is still quadrilateral. However, the Catmull-Clark subdivision scheme depends on every vertex having exactly four neighbours for the generation of the new vertex that is near to the old vertex position (the rightmost case in Figure 4). Catmull and Clark got around this problem by creating a new set of weights, one set of weights for each vertex valence (the valence of vertex is a number of other vertices to which it is connected). Instead of weights of $1/64$, $6/64$, and $36/64$ you can use weights of $1/4n^2$, $3/2n^2$, and $1 - 7/4n$, where n is the valence of the vertex. These are Dennis Zorin's weights for extraordinary vertices, other values can also be used.

However, this is not the only type of mesh with which we can deal. The Doo-Sabin scheme can be easily modified to cope with meshes in which some of the polygons are not quadrilateral, while still maintaining C^1-continuity everywhere. For a k-sided polygon, the weights, α_k on the k vertices can be shown to be:

$$\alpha_0 = \frac{1}{4} + \frac{5}{4k} \tag{118}$$

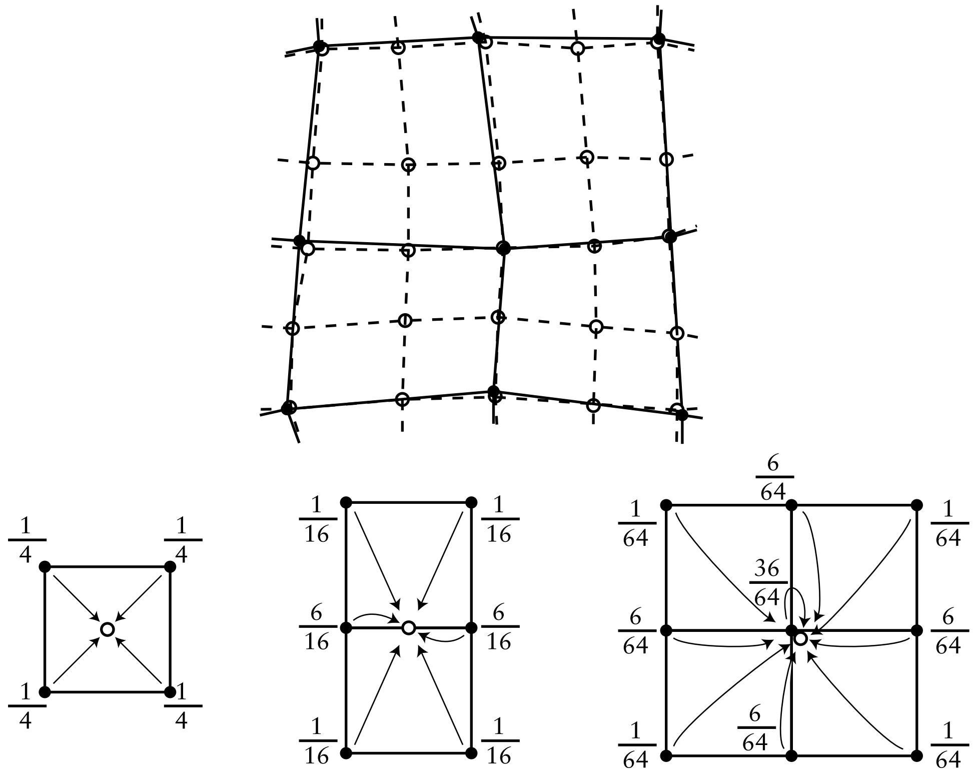


Figure 4: Catmull-Clark subdivision. Above a mesh (solid dots and solid lines) that has been refined (open dots and dashed lines). Below the weights used to generated each type of refined vertex: centre, edge, and modified old vertex.

$$\alpha_i \quad = \quad \frac{1}{4k}\left(3 + 2\cos\frac{2i\pi}{k}\right) \tag{119}$$

Other schemes, notably the Butterfly, Loop (named after Dr Loop), and $\sqrt{3}$ schemes handle triangular meshes. Some of these may be discussed in the lectures.