Mathematics of 3D Graphics

Juan David González Cobas
Universidad de Oviedo
`cobas@epsig.uniovi.es`

Blender Conference ’2004

Contents

1 Coordinate systems **2**

2 Vector algebra **4**
  2.1 Vector operations 4
  2.2 Geometric interpretations 4
  2.3 Bases 6

3 Affine and projective geometry **6**
  3.1 Planes 7
  3.2 Lines 8
  3.3 Incidence and metric geometry 9

4 Matrices and geometric transforms **9**
  4.1 Projective recap 14
  4.2 Determinants 16

5 Numerical linear algebra and equation solving **17**
  5.1 The LU decomposition 18
  5.2 Computing determinants 19
  5.3 Solving linear equation systems 19
  5.4 Matrix inversion 20
  5.5 Equation solving 20

6 Differential geometry in a nutshell **21**
  6.1 Curves 21
  6.2 Surfaces 23

7 Interpolation and approximation **25**
  7.1 Lagrange interpolation formula 25
  7.2 Least squares fitting 27

8 Modelling of curves and surfaces: splines 28
8.1 Natural splines 28
8.2 Hermite interpolants 29
8.3 Bézier curves 29
8.4 B-splines 31
8.5 NURBS 32
8.6 Splines for parametric surfaces 32

Abstract

A math refresher for the tasks 3D artists and developers face every day, including aspects of the geomety of 3D graphics, 3D object representation and geometric transformations, to wit:

- Coordinate systems
- Elementary algorithms (3D affine geometry of lines, planes, distance and intersections)
- Vector wizardry (math tricks to speed up things)
- Geometric transforms (rotation, dilation, shear)
- Matrices and some numerical linear algebra (equation solving)
- Interpolation and approximation (splines, NURBS and their relatives)
- Differential geometry of surfaces in a nutshell (coordinate patches, volume forms and the dreadful Jacobian)
- Whatever you want to know and never dared to ask about 3D math
- Antialiasing (only for render fans)

1 Coordinate systems

Three-dimensional scene description requires mainly using a 3D cartesian coordinate system. Points in space are uniquely determined by their three *cartesian coordinates* (x, y, z).

Other coordinate systems are not so frequently used in 3D graphics. The most common are *cylindrical* and *spherical* coordinates.

The three systems label points with coordinates according to the schema of figure 1. Each system uses three of the dimensions referred thereto:

cartesian: (x, y, z)
cylindrical: (ρ, ϕ, z)
spherical: (r, θ, ϕ)

The equations allowing us to switch from and to other coordinate systems are

Cartesian ↔ Spherical	**Cartesian ↔ Cylindrical**
$x = r\sin\theta\cos\phi$	$x = \rho\cos\phi$
$y = r\sin\theta\sin\phi$	$y = \rho\sin\phi$
$z = r\cos\theta$	$z = z$

Of greater importance for computer graphics is the usage of *homogeneous or projective coordinates*. Ordinary points in space are given *four* coordinates instead of three:

$$(x, y, z) \leftrightarrow (x, y, z, w)$$

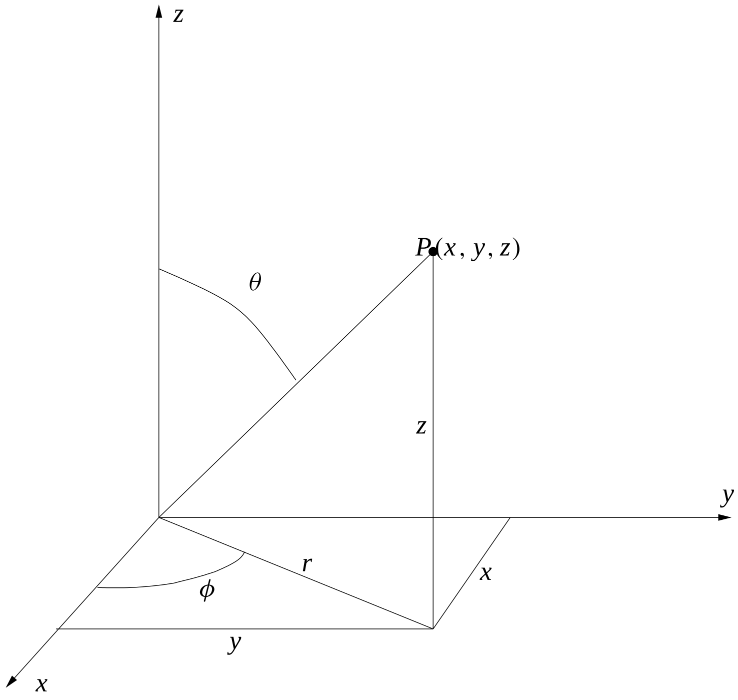


Figure 1: Cartesian, cylindrical and spherical coordinate systems

This introduces an obvious redundancy, so that the same point in 3D has infinitely many homogeneous coordinates, according to the equivalence

$$\begin{gathered}(x, y, z, w) \equiv (x', y', z', w') \Leftrightarrow \\ \Leftrightarrow \alpha(x, y, z, w) = (x', y', z', w') \qquad \text{for some } \alpha \neq 0\end{gathered}$$

so that proportional 4-tuples denote the same point. The usual (x, y, z) triple is identified with the $(x, y, z, 1)$ quadruple, and the quadruple (x, y, z, w) denotes a point $(x/w, y/w, z/w)$ in ordinary space.

If $w = 0$, we have a *point at infinity*; what we actually get with this 4-tuple playing is a coordinatization of *projective 3-space* geometry. This is the first advantage of this representation. No special treatment of points at infinity or parallelism is needed. Projective transformations are easily transliterated into linear transformations in 4-tuple space (that is, matrices). Another advantage is that the usual affine transformations get also a regular treatment as matrix products, as we shall see in 4

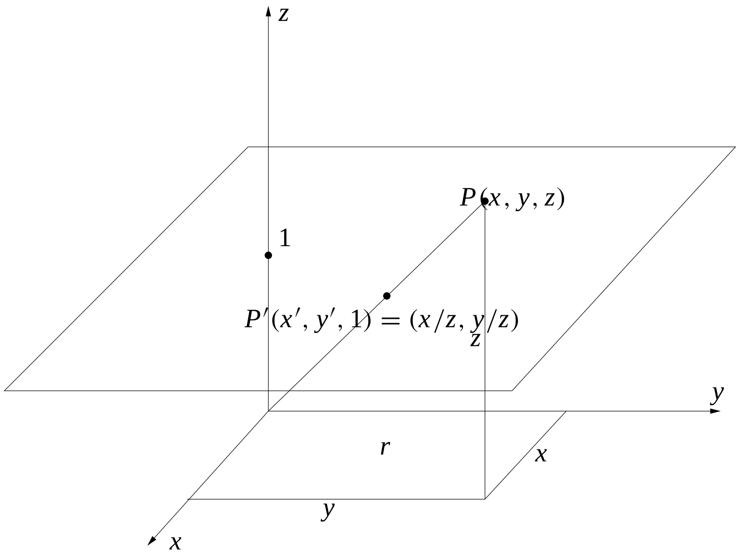


Figure 2: Projective coordinates

2 Vector algebra

2.1 Vector operations

Vectors in 3D space are usually given by their cartesian coordinates, and operations on vector can be defined in terms of them:

addition $(x, y, z) + (x', y', z') = (x + x', y + y', z + z')$

substraction $(x, y, z) - (x', y', z') = (x - x', y - y', z - z')$

scaling $\lambda(x, y, z) = (\lambda x, \lambda y, \lambda z)$

dot product $(x, y, z) \cdot (x', y', z') = xx' + yy' + zz'$

norm $\|(x, y, z)\| = \sqrt{v \cdot v} = \sqrt{x^2 + y^2 + z^2}$

cross product $(x, y, z) \times (x', y', z') = (x, y, z) \wedge (x', y', z') = (yz' - zy', zx' - xz', xy' - yx')$

2.2 Geometric interpretations

The geometric interpretation of these operations can be seen in figure 3. We can see that the dot product has the alternative interpretation as

$$v \cdot w = \|v\| \, \|w\| \cos \phi$$

and the cross product is orthogonal to the factors and has modulus equal to the area of the parallelogram defined by the factors:

$$v \times w = \|v\| \, \|w\| \sin \phi$$

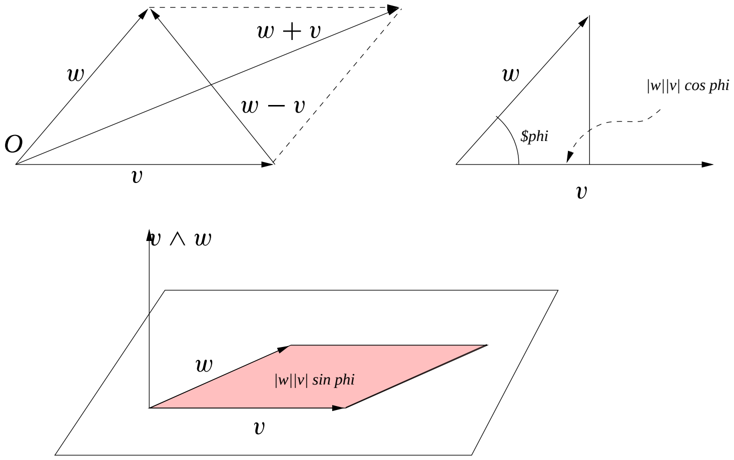
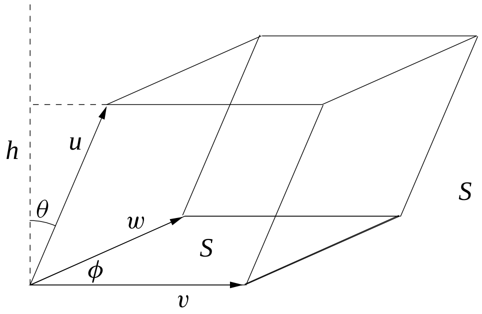


Figure 3: Geometrical interpretation of vector operations



$$V = S\|u\|\cos\theta$$
$$S = \|v\|\|w\|\sin\phi = |vxw|$$

V = u dot (v x w) = det (u, v, w)

Figure 4: Volume of a cell

From them, we infer that the two products deserve special attention, because they allow computing the metric elements of a model: distances, angles, surfaces, volumes, orthogonality and collinearity. Some examples are in order:

- The norm of a vector $v = (x, y, z)$ (its length) is computed through scalar product: $\|v\| = \sqrt{(}v \cdot v)$
- The distance between two points is the norm of the vector joining them:

$$\begin{aligned} P &= (x, y, z) \\ Q &= (x', y', z') \\ PQ &= (x' - x, y' - y, z' - z) \\ d(P, Q) = \|PQ\| &= \sqrt{(x' - x)^2 + (y' - y)^2 + (z' - z)^2 +} \end{aligned}$$

- The angle ϕ between v and w is

$$\cos\phi = \frac{v \cdot w}{\|v\|\,\|w\|}$$

- Two vectors are orthogonal iff $v \cdot w = 0$
- Two vectors are collinear iff $v \times w = 0$
- A normal vector to the plane defined by v and w is given by $v \times w$
- The volume of the cell defined by the vectors u, v, w is (see figure 4)

  $$\det(u, v, w) = u \cdot (v \times w) = \begin{vmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix}$$

  This is known as the *triple product* of the three vectors, or, more mundanely, as their *determinant*. Its computation follows well-known rules which we will not delve in.

2.3 Bases

Given any set of vectors $\{v_1, v_2, \ldots, v_n\}$, its *span* is the set of all vectors we can construct as linear combinations of them (using scaling and addition):

$$< v_1, v_2, \ldots, v_n >= \{w | w = \lambda_1 v_1 + \lambda_2 v_2 + \cdots + \lambda_n v_n\})$$

The set $\{v_1, v_2, \ldots, v_n\}$ is *free*, or *linearly independent*, if

$$\lambda_1 v_1 + \lambda_2 v_2 + \cdots + \lambda_n v_n = 0 \Leftarrow \lambda_1 = \lambda_2 = \cdots = \lambda_n = 0$$

or, more mundanely, if no vector of the set can be expressed as a combination of the others. A free set which spans the whole space in consideration is a *basis* for it. In our ordinary 3-space, all bases have three elements e_1, e_2, e_3. Moreover, a basis where $e_i \cdot e_j = 0$ if $i \neq$ is said to be *orthogonal*; in addition, $e_i \cdot e_i = 1$ makes this base *orthonormal*.

In an orthonormal base, every vector has a simple and convenient expansion

$$x = (x \cdot e_1)e_1 + (x \cdot e_2)e_2 + (x \cdot e_3)e_3$$

3 Affine and projective geometry

In the previous section on vectors we made a passing mention to the distance between points, and made liberal use of the notation PQ to mean the vector joining P to Q. Strictly speaking, this is only legitimate if ordinary 3-space (without coordinates) is given a structure known as *affine space*. This simply means we are given a way to translate any point P by any vector v, the result being written as $P + v$. The translation $+$ must satisfy the obvious conditions

1. For every vector v, $P \to P + v$ is a 1-to-1 correspondence
2. For every point P, $P + 0 = P$
3. For every point P and vectors v, w, $(P + v) + w = P + (v + w)$

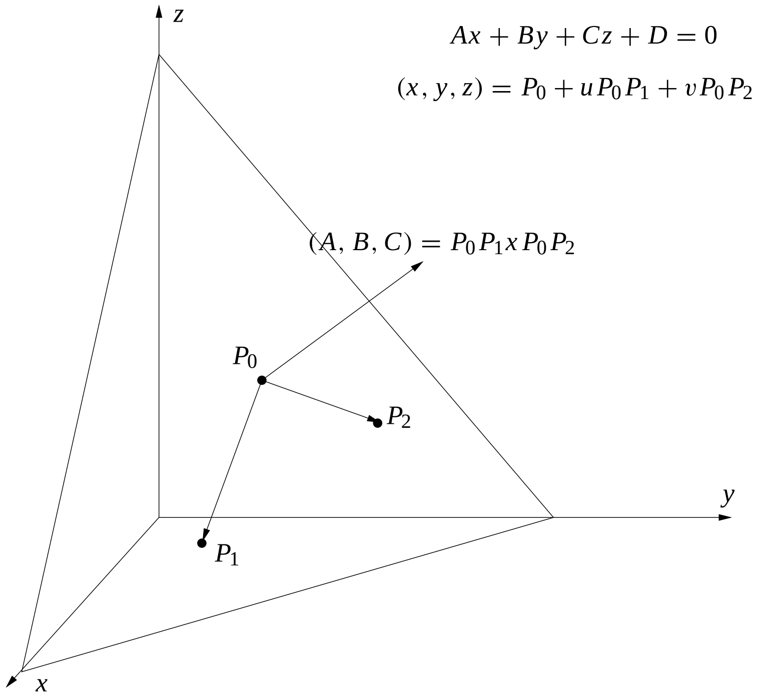

z
$Ax + By + Cz + D = 0$
$(x, y, z) = P_0 + u P_0 P_1 + v P_0 P_2$
$(A, B, C) = P_0 P_1 x P_0 P_2$
P_0
P_2
y
P_1
x

Figure 5: Equations of a plane

This properties are so customary in ordinary 3D work that we use them without even being aware. The existence of a vector PQ joining P to Q is a consequence of them; the usual coordinatization of space is another. Adding to this the existence of a scalar product of vectors, concepts of distance, angles and orthogonality can be defined, and *every* problem involving metric geometry can be solved analytically.

We give a short account of the usual *straight* entities in 3D affine geometry: points, lines, planes. These are the *affine* or *linear manifolds* of standard geometry, and can be defined by systems of linear equations on coordinates.

3.1 Planes

A plane can usually defined in two ways

implicit equation By means of a single linear equation which the points in the plane satisfy

$$Ax + By + Cz + D = 0$$

parametric equation The points of the plane are obtained by sweeping all the possible values of two real parameters

$$P = P_0 + u\ P_0 P_1 + v\ P_0 P_2$$

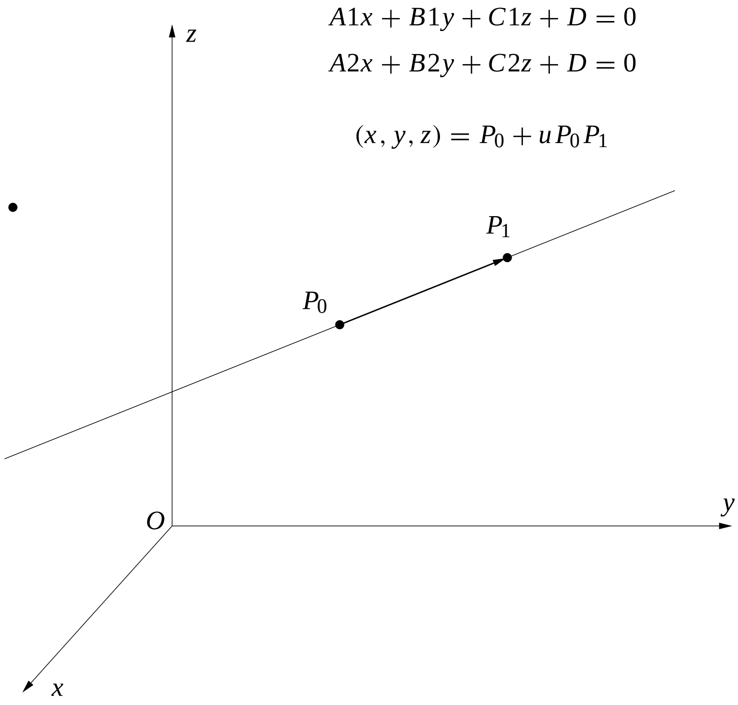

$A1x + B1y + C1z + D = 0$
$A2x + B2y + C2z + D = 0$
$(x, y, z) = P_0 + uP_0P_1$
z
P_1
P_0
O
y
x

Figure 6: Equations of a line

or, in coordinates

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + u \begin{pmatrix} a \\ b \\ c \end{pmatrix} + v \begin{pmatrix} a' \\ b' \\ c' \end{pmatrix}$$

The vectors (a, b, c) and (a', b', c') are *direction vectors* of the plane.

Converting between both representations is quite easy: from the implicit equations it is easy to obtain three (non-collinear) points P_0, P_1, P_2 giving the parametric equations. Passing from parametric to implicit would require ellimination of the u, v parameters. But there is a faster way:

$$\begin{aligned} (A, B, C) &= P_0P_1 \times P_0P_2 = (a, b, c) \times (a', b', c') \\ D &= -(A, B, C) \cdot P_0 \end{aligned}$$

3.2 Lines

Straight lines can be defined by a point and a direction, or by the intersection of two planes. This gives rise to the possible representations of lines with

implicit equations Two (independent) linear equations givin the line as the intersection of two planes

$$\begin{aligned} A_1x + B_1y + C_1z + D_1 = 0 \\ A_2x + B_2y + C_2z + D_2 = 0 \end{aligned}$$

parametric equation Giving the coordinates of points of the line as a parameter u sweeps all real values

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = P_0 + uP_0P_1 = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + u \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Again, converting between both representations is easy: the parametric equation can be put in implicit form by ellimination of the parameter; and from the implicit form, we get easily two points, or one point and the direction vector

$$P_0P_1 = (A_1, B_1, C_1) \times (A_2, B_2, C_2)$$

3.3 Incidence and metric geometry

With these concepts, and the properties of dot and cross product in mind, we can solve any problem of elementary 3D analytical geometry. Better than enumerating a boring list of different cases, we prefer to state some sample problems to give a feeling of the techniques involved.

For instance, we want to compute the distance between two lines, as shown in figure 3.3. Both lines r_1 and r_2 come defined by points P_i and unit direction vectors u_i. A moment's thought shows that the minimum distance d between two disjoint lines is realized between points Q_1 and Q_2 such that the segment Q_1Q_2 is orthogonal to both r_1 and r_2. To see this more clearly, there are two parallel planes π_1 asnd pi_2 containing each line, and Q_1Q_2 is perpendicular to them. So

$$d = \|Q_1Q_2\| \qquad \text{and} \qquad Q_1Q_2 \perp u_1, u_2$$

So $Q_1Q_2 = du_1 \times u_2$. Now, vector P_1P_2 joins points of both lines orthogonal to Q_1Q_2. Then, the dot product with unitary $u_1 \times u_2$ gives the magnitude of the projection of P_1P_2 on Q_1Q_2, which is exactly d. We get

$$d = P_1P_2 \cdot u_1 \times u_2 = \det(P_1P_2, u_1, u_2)$$

4 Matrices and geometric transforms

The tasks we can perform until now are quite elementary. For a much powerful management of geometric objects, we would like to express more complex operations on shapes. Matrices are the standard way of representing linear operations and computing their action.

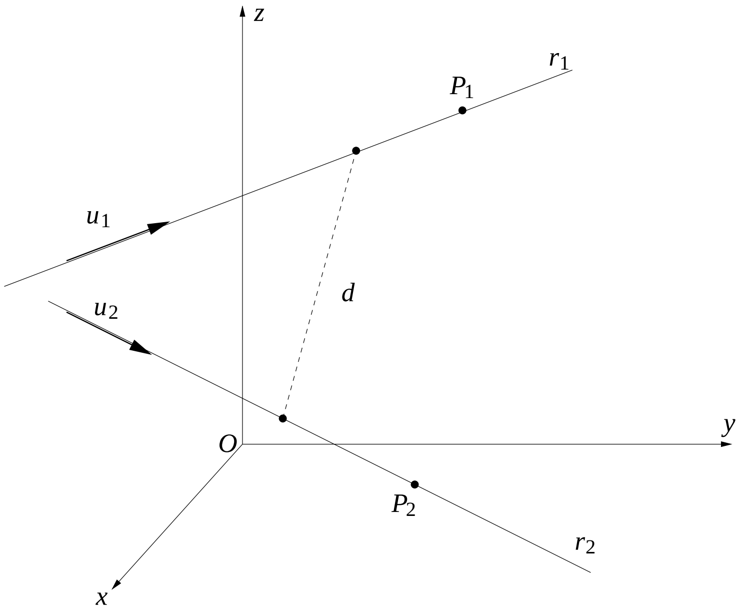


Figure 7: Example of the distance between two lines

An $m \times n$ matrix is a rectangular array of numbers like this

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

which is more compactly written as $(a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$ or simply as (a_{ij}) if dimensions are understood.

Like vectors, matrices can be added and subtracted componentwise. The product of matrices is given by the rule

$$\begin{array}{rl} A = (a_{ij}) & \qquad i = 1 \dots m,\ j = 1 \dots n \\ B = (b_{jk}) & \qquad j = 1 \dots n, k = 1 \dots p \\ AB = (c_{ik}) & \qquad i = 1 \dots m, k = 1 \dots p \end{array}$$

and

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

Note that, even when $m = n = p$, we have $AB = BA$, that is, the product of matrices is noncommutative.

The transpose of a matrix is obtained by exchanging rows and columns

$$A = (a_{ij}) \qquad \Rightarrow \qquad A^T = (a_{ji})$$

A $n \times n$ matrix A is *regular* if there is another $n \times n$ matrix such that $AB = BA = 1$. We usually write A^{-1} for the inverse of A.

We will usually restrict ourselves to *row vectors* and *column vectors* to refer to points and vectors, and *square matrices* (with $m = n$) to represent linear transforms. The dimensions of our matrices will be 3×3 or 4×4 almost exclusively.

Let us start by a *translation* of (the points of) an object by a vector $v = (v_x, v_y, v_z)$. This transformation sends a point (x, y, z) to (x', y', z'), where

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{1}$$

or, more compactly, $P' = P + v$. We see here that this transformation is not, strictly speaking, linear: it has independent terms.

What if we want to express analytically a mirror reflection around the XY-plane? This has the property of changing signs of z-coordinates leaving everything else unchanged. We need to prescribe these independent coordinate changes by means of independent equations

$$\begin{aligned} x' &= x \\ y' &= y \\ z' &= -z \end{aligned}$$

which lead to

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

As a matter of fact, *any* linear transformation can be put in matrix form. The recipe is:

1. take a basis e_1, e_2, e_3
2. apply the transform to the vectors, obtaining $f_1 = Te_1,\ f_2 = Te_2,\ f_3 = Te_3$
3. put the vectors obtained as columns of the matrix.

Another standard transformation is a *scaling* by a factor λ. Of course, if $0 < \lambda < 1$ this is better called a *contraction*, while if $\lambda > 1$ we have a *dilation*. And, for $\lambda = -1$, we get a *reflection* around the origin. We have that every coordinate of a point gets scaled by λ, so we put

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{2}$$

A more interesting case is a *rotation*, determined by the axis and the angle of rotation. A rotation of angle ϕ around the z-axis can be computed using the recipe

1. take as base vectors $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$.
2. rotate them, obtaining $(\cos\phi, \sin\phi, 0)$, $(-\sin\phi, \cos\phi, 0)$, $(0, 0, 1)$
3. put them as columns of the matrix, resulting in

$$R_\phi = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

We can check that this is the correct transformation of coordinates:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3}$$

If we rotate an angle θ around the x-axis, we get a different matrix

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4}$$

And, if we apply both rotations in order of appearance, we get

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{5}$$

The composition of both rotations is given by the *product matrix*, which in this case is

$$R = R_x R_z = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \cos\theta\sin\phi & \cos\phi\cos\theta & -\sin\theta \\ \sin\phi\sin\theta & \cos\phi\sin\theta & \cos\theta \end{pmatrix}$$

This does not look like any of the rotations seen before, but it *is* indeed a rotation around some axis. How do we know? Because the resultant matrix is orthogonal:

$$R^t R = R R^t = 1$$

where 1 denotes the identity matrix having 0's outside the main diagonal and 1's in it. Such a matrix represents a length-preserving transformation.

A *shear* does not preserve lengths. An example of shear is given by

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{6}$$

This has the effect of shifting the "upper layers" of a cube in a fashion similar to a sheared deck of cards, as seen in figure 8.

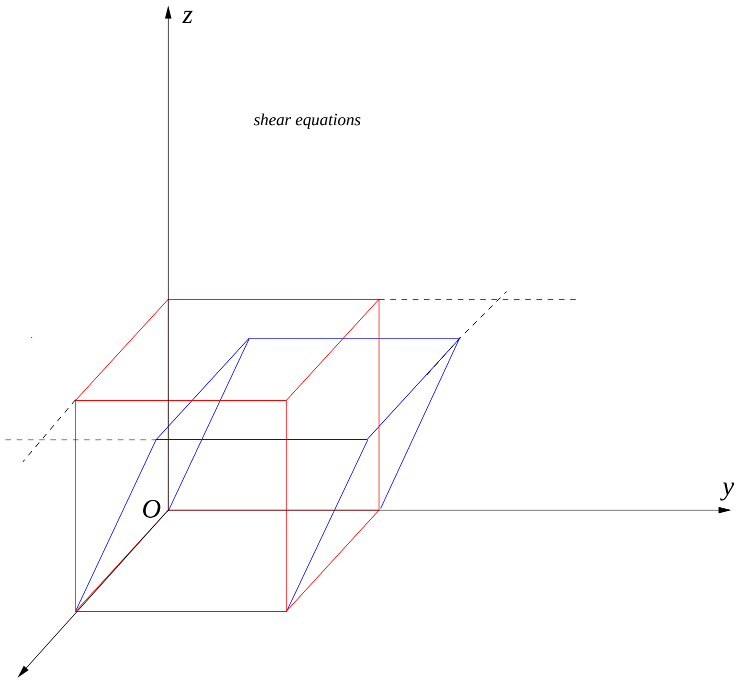

z
shear equations
y
O

Figure 8: Shearing a cube

What if we want to compose a shear with a translation, followed by a rotation? It gets a bit messy:

$$X' = R(SX + B) = RSX + RB$$

because of the independent terms in the middle of the product. In general, we will obtain a transformation of the type

$$X' = AX + B \qquad \text{or} \qquad \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{7}$$

Now we can appreciate the advantage of using homogenoeus coordinates. The ordinary affine space can be coordinatized with a unit fourth component, so that equation 7 can be written as

$$X' = AX \qquad \text{or} \qquad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{8}$$

All of the preceding transforms can be written this way, and any affine transformation takes, in homogeneous coordinates, the form of a linear transformation, and composes by matrix multiplication.

4.1 Projective recap

As a recapitulation, let us see the form that some transformations take in projective 4-coordinates (see figure 4.1).

Translation

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflection

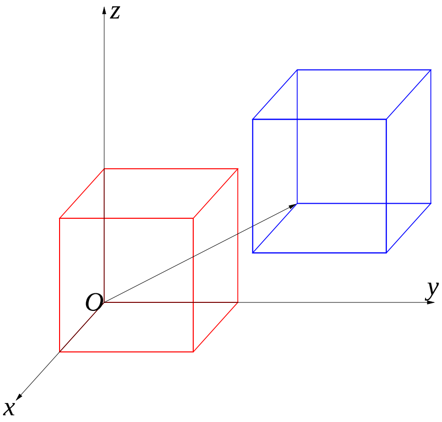
$$F = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Shear

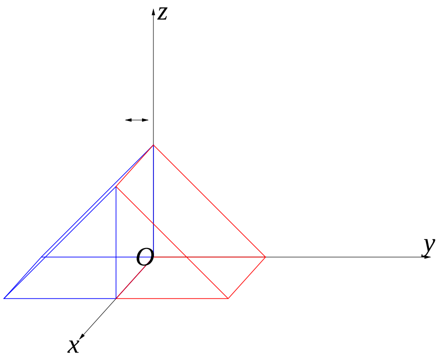
$$S = \begin{pmatrix} 1 & 0 & s_x & 0 \\ 0 & 1 & s_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation

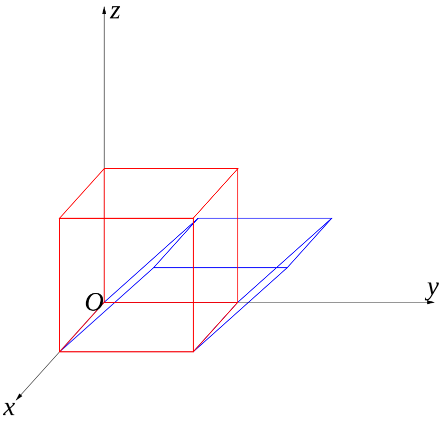
$$S = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



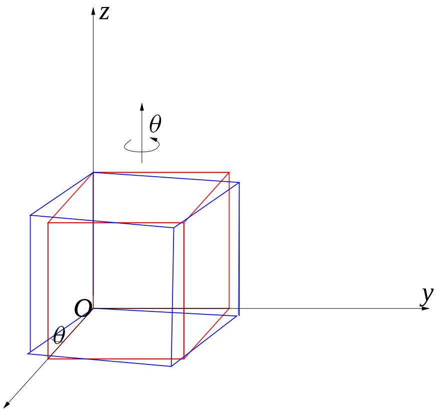
(a) Translation



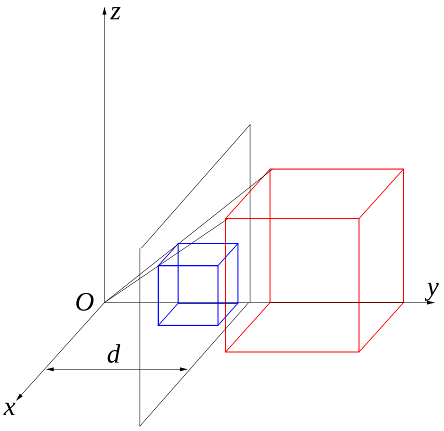
(b) Reflection



(c) Shear



(d) Rotation



(e) Projection

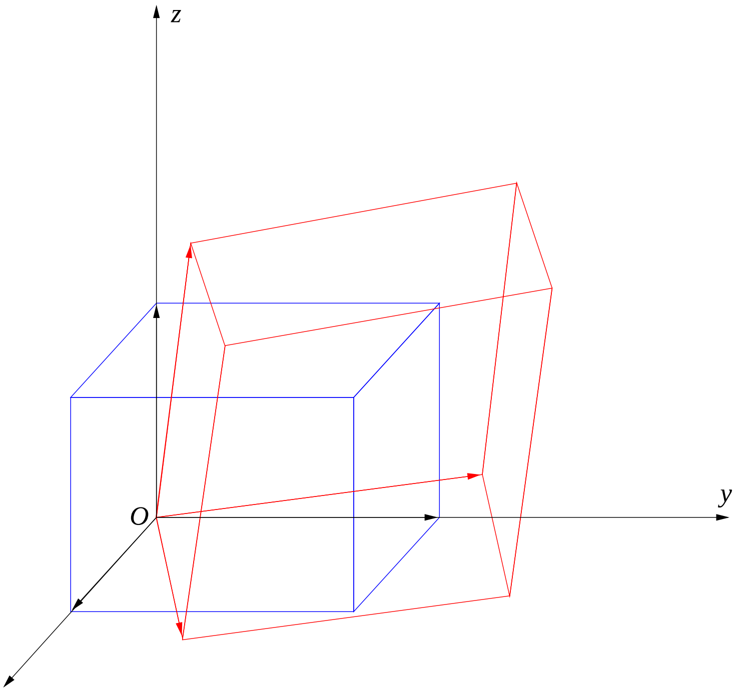


Figure 9: Determinant as a volume measure

Projection

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1/d & 0 & 0 \end{pmatrix}$$

4.2 Determinants

Let us consider an arbitary linear transformation like depicted in figure 9. The blue box becomes distorted by it, becoming a blue parallelepiped. We have made the edges on the axes fit the base vectors e_1, e_2, e_3, so that the edges corresponding in blue are the transformed vectors $f_1 = Te_1,\ f_2 = Te_2,\ f_3 = Te_3$.

What is the volume of the blue box? Three facts are obvious

- If two vectors are the same, the volume is zero

$$V(f_1,\ f_2,\ f_3) = 0 \text{ if } f_1 = f_2$$

- A shear does not change volume (check figure 8).

$$V(f_1 + f_2,\ f_2,\ f_3) = V(f_1,\ f_2,\ f_3)$$

- Scaling a vector scales the volume.

$$V(\lambda f_1,\ f_2,\ f_3) = \lambda V(f_1,\ f_2,\ f_3)$$

and the like properties are valid for any other functional position of $V(\cdot)$. These happen to be the properties of the *determinant* of a matrix, which is defined uniquely (except for a constant factor) by the properties above. We write

$$V(f_1,\ f_2,\ f_3) = \begin{vmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{vmatrix} = \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \prod_{i=1}^{n} a_{i\pi(i)} \tag{9}$$

The general definition of a determinant, given as third term in formula 9, is of mainly theoretical interest and gives the value of a determinant for small matrices:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = +a_{11}a_{22} - a_{12}a_{21}$$

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = + a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} \\ - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31}$$

A rearrangement of the definition (9) gives the formula

$$\det(A) = \sum_{i=1}^{n} (-1)^{i-1} det(A_{1i}) \tag{10}$$

where A_{1i} is A after deletion of its first row and i-th column. This allows recursive computation of small determinants:

$$\begin{vmatrix} 6 & 3 & 9 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{vmatrix} = 6 \begin{vmatrix} 3 & 1 \\ 2 & 3 \end{vmatrix} - 3 \begin{vmatrix} 2 & 1 \\ 1 & 3 \end{vmatrix} + 9 \begin{vmatrix} 2 & 3 \\ 1 & 2 \end{vmatrix} \\ = 6(9-2) - 3(6-1) + 9(4-3) = 36$$

For higher orders, this is of no use, and higher-order determinants are better computed by using the defining properties of the function, to wit: a linear combination of any row can be added to any other row. This allow making zeroes in a column so that the computation is reduced to a one-less-order determinant by virtue of the expansion (10). A much faster way of applying this recipe is given in the next section.

5 Numerical linear algebra and equation solving

In section 4, we found out that some tasks are quite complex computationally. We would like to be alleviated of this burden to avoid our programs sink in eternal loops. Too often we need

- inverting a matrix
- computing a determinant
- solving a system of linear equations
- finding an orthogonal/orthonormal basis of a linear space
- finding the roots of a polynomial
- finding the solutions of nonlinear equations, or systems of them
- fitting a curve or function to a set of points or other data

Most of these tasks are the subject of *numerical linear algebra*; we do not care about the algebraic properties of linear operations, matrices and determinants, but just like to find a solution, a correct solution, and by a fast method. Tasks not involving matrices in the above list are the subject of *numerical analysis* in general; again, no fancy mathematical properties are of interest, but only solving things, fast, and accurately.

5.1 The LU decomposition

This is a beautified form of what we usually call *Gaussian elimination*: the process of making zeroes in a matrix by linearly combining rows. This idea takes us eventually to a matrix in upper triangular form (the U part). If we also reckon the operations performed, applying them to a blank, brand new unit matrix, we get a lower triangular matrix (the L part of the beast), and, magically, we come to something like this

$$A = \begin{pmatrix} 1 & 6 & 4 \\ 2 & 3 & 5 \\ 8 & 2 & 3 \end{pmatrix} \to \begin{pmatrix} 1 & 6 & 4 \\ 0 & -9 & -3 \\ 0 & -46 & -29 \end{pmatrix} \to \begin{pmatrix} 1 & 6 & 4 \\ 0 & -9 & -3 \\ 0 & 0 & -41/3 \end{pmatrix} = U$$

Recording the multipliers in a blank unit matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 8 & 0 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 8 & 46/9 & 1 \end{pmatrix} = L$$

and, magic:

$$LU = A$$

The procedure can even be performed in place. The usual library function can be found in any respectable linear algebra package, and gives, usually, a matrix with the L and U parts merged in one, and a vector of permutations of rows, used in pivoting for numerical stability. Then, LU decomposition becomes

$$PA = LU$$

where P is a permutation matrix whose effect is permuting the rows of A.

For example, feeding the LU routine of LINPACK with

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 4 & -1 \\ 2 & 1 & 5 \end{pmatrix}$$

we get

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \qquad L = \begin{pmatrix} 1.00 & 0.00 & 0.00 \\ 0.50 & 1.00 & 0.00 \\ 0.50 & 0.42 & 1.00 \end{pmatrix} \qquad U = \begin{pmatrix} 2.00 & 1.00 & 5.00 \\ 0.00 & 3.50 & -3.50 \\ 0.00 & 0.00 & 2.00 \end{pmatrix}$$

This decomposition has three main uses

5.2 Computing determinants

We get

$$\det(A) = \det(P)\det(L)\det(U)$$

A permutation matrix has determinant ±1 according to the parity of the permutation. The lower triangular matrix has obviously $\det(L) = 1$ and we get that, up to a sign, $\det(A)$ is the product of U's main diagonal. This allows computing determinants in time $O(n^3)$, which is the usual complexity of the LU decomposition.

5.3 Solving linear equation systems

Now suppose we have a system of linear equations like this

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

better written in matrix form as

$$AX = B \qquad \text{or} \qquad \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

We can use the LU decomposition to advantage:

$$AX = P^{-1}LUX = B \qquad \text{or} \qquad LUX = PB \tag{11}$$

Except for a permutation of independent terms, we can solve equation (11) by forward substitution and backsubstitution, because from

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

we obtain

$$w_1 = b_1 \tag{12}$$
$$w_2 = b_2 - l_{21} w_1 \tag{13}$$
$$w_3 = b_3 - l_{31} w_1 - l_{32} w_2 \tag{14}$$

and, in general

$$w_k = b_k - \sum_{j=1}^{k-1} l_{kj} w_j \tag{15}$$

Once we know w_i, we do the same trick backwards with U. From

$$\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \dots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix}$$

we write

$$x_n = (1/u_{nn}) w_n \tag{16}$$
$$x_{n-1} = (1/u_{n-1,n-1})(w_{n-1} - u_{n-1,n} x_n) \tag{17}$$
$$x_{n-2} = (1/u_{n-2,n-2})(w_{n-2} - u_{n-2,n} x_n - u_{n-2,n-1} x_{n-1}) \tag{18}$$

and, in general

$$x_k = (1/u_k)(x_k - \sum_{j=n}^{k+1} u_{k,j} w_j \tag{19}$$

and the system is solved.

Note that if a system has to be solved with various sets of b values, the LU decomposition can be reused, and forwad- and backward substitution is a $O(n^2)$ process. This has direct application to

5.4 Matrix inversion

Applying the processes (12) and (16) to the columns of the identity, we get the columns of the inverse of A. This is the most effective inversion procedure in practice for non-sparse matrices.

5.5 Equation solving

What happens when we need to solve a nonlinear equation? Suppose this: you have a nice surface modelled with a function $P(u, v)$ that gives the position vector of a point in the surface as a function of the parameters u, v. And now you are asked to find the intersection of the surface with the Z-axis.

The intersection would be found by solving

$$\begin{aligned} 0 &= P_x(u, v) \\ 0 &= P_y(u, v) \\ z &= P_z(u, v) \end{aligned}$$

for u, v, z. Think, for instance, that P is a NURBS surface. What can we do?

We better put the system in the form

$$f(u, v, z) = 0 \qquad \text{with} \qquad f(u, v, z) = \big(P_x(u, v) P_y(u, v) P_z(u, v) - z\big)$$

In general, this is a difficult problem to be solved by a numerical (iterative) procedure. Let us try *Newton's method*. We start with an initial value of $(u, v, z) = (u_0, v_0, z_0)$, and we refine it repeatedly by applying the formula

$$(u_{i+1}, v_{i+1}, z_{i+1}) = (u_i, v_i, z_i) - Df^{-1}(u_i, v_i, z_i)\, f(u_i, v_i, z_i)$$

Here, $Df(u_i, v_i, z_i)$ is the matrix of partial derivatives

$$\begin{pmatrix} \frac{\partial f_x}{\partial x} & \frac{\partial f_x}{\partial y} & \frac{\partial f_x}{\partial z} \\ \frac{\partial f_y}{\partial x} & \frac{\partial f_y}{\partial y} & \frac{\partial f_y}{\partial z} \\ \frac{\partial f_z}{\partial x} & \frac{\partial f_z}{\partial y} & \frac{\partial f_z}{\partial z} \end{pmatrix}$$

known as the *Jacobian of* f evaluated at u_i, v_i, z_i. If you are asking, yes, this is the same jacobian we will find later in section 8.6. If you are lucky, the succesive points come closer to a solution of the equation after a few iterations. If you are not…

Newton's method has quadratic convergence if you happen to start close enough to a solution. This means that you double the number of exact digits of your solution if things go properly (which, of course, does not happen when you need).

6 Differential geometry in a nutshell

6.1 Curves

A curve in space is given by a parametric representation

$$t \to \phi(t)$$

where $\phi(t)$ will be a generic point of the curve, which is swept by varying t within an interval (the *domain* of the parameter).

The speed at which we go through the curve is given by $\phi'(t)$, and acceleration by $\phi''(t)$. The length of a segment of curve is given by the formula

$$L(t_0, t_1) = \int_{t_0}^{t_1} \sqrt{1 + \phi'(t)^2}\, dt$$

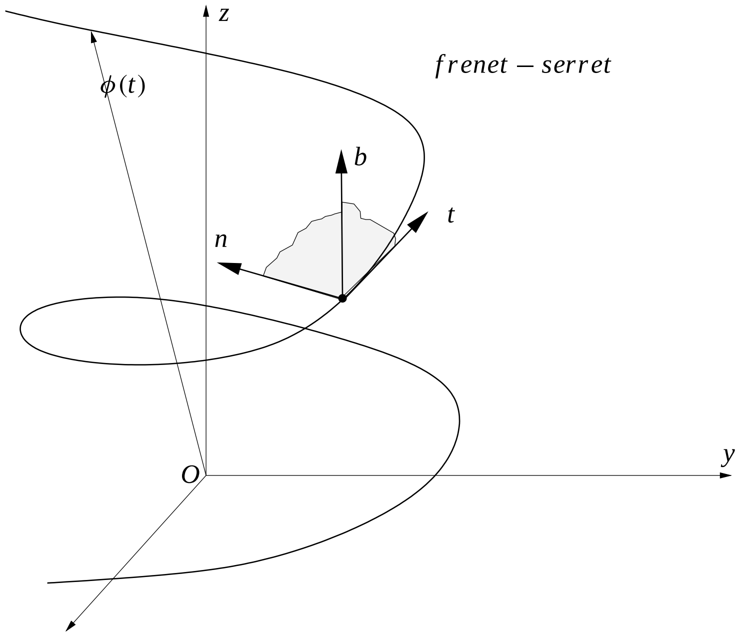


Figure 10: Frenet moving frame

If we put $s = L(t_0, t)$, we may use s as the parameter. This has the advantage to make computations a bit simpler.

The decomposition

$$\phi'(t) = \frac{d\phi}{dt} = \|\phi'(t)\|T = vT$$

becomes simply

$$\phi'(s) = \frac{d\phi}{ds} = \|\phi'(s)\|T = T$$

where T is a unit tangent vector. As $T \cdot T$ is a constant, $2T' \cdot T = 0$ and we conclude that T' is orthogonal to T. So it is a normal vector whose module measures the rotation speed of T, inversely proportional to the radius of curvature. We call it the *curvature* κ

$$\phi''(s) = T' = \kappa N$$

with N a unit normal vector. Putting $B = T \times N$ it is easy to derive that the orthonormal triad (T, N, B), a function of parameter s named the $Frenet frame$, satisfies

$$\begin{aligned} T' &= \kappa N \\ N' &= -\kappa T - \tau B \\ B' &= \tau N \end{aligned} \tag{20}$$

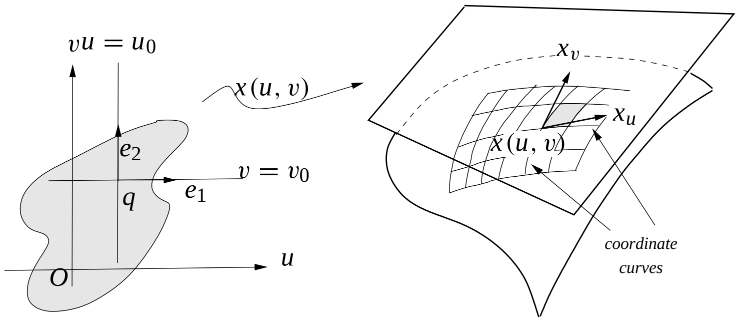


Figure 11: Parametric surface

were t is a function that measures how much the curve takes off being planar: the *torsion*. We call B the *binormal* vector, and equations (20) are the *Frenet-Serret* formulas

Note for physicists: surely you noticed that

$$T' = D \times T$$
$$N' = D \times N$$
$$B' = D \times B$$

with $D = \kappa B - \tau T$. Would you prefer to say $D = \Omega$?

6.2 Surfaces

We will see in section 8 that, apart from triangle or polygon meshes, surfaces can be represented parametrically by more complex functions like bicubic polynomials. In general, we may describe a surface by means of a *coordinate patch*: a vector function of two parameters (u, v) which gives points of the surface as (u, v) sweep their domain of values, as shown in figure 11

It is of interest to know how to make computations with this representation. We can see in the figure that, keeping v fixed and moving u, we obtain a family of curves, parametrized by the fixed v value; exchanging coordinates another family is obtained, and we get a network of *coordinate curves* which cover the whole surface patch.

The vector $x(u, v)$ is a function of two variables. Differentiating partially we get two *tangent vectors*

$$x_u = \frac{\partial x}{\partial u}$$
$$x_v = \frac{\partial x}{\partial v}$$

which, togethet with the point $x_0 = x(u_0, v_0)$, define the *tangent plane* at that point. The normal at x_0 is obtained in the usual way

$$N = x_u \times x_v \tag{21}$$

and can be normalized if necessary.

What about non-coordinate curves? Let us suppose we have a curve in parametrical form $t \to \phi(t)$. If it remains in the surface, it should be possible to write

$$\phi(t) = x(u(t), v(t))$$

The tangent vector to the curve at any point is given by differentiating ϕ

$$\phi'(t) = \frac{\partial x}{\partial u} u'(t) + \frac{\partial x}{\partial v} v'(t) = x_u u'(t) + x_v v'(t)$$

We can see that these tangent vectors play an important role: they span the set of *all* tangent vectors at the given point x_0, and they form, indeed, a basis of the tangent plane. But there is more to it. Look at the shaded rhomb in figure 11. Ignoring curvature, it is the image of a coordinate rectangle of unit area. What is the area of this surface element? We know many ways of computing it now. For instance

$$\text{Area} = \|x_u \times x_v\|$$

This magnitude measures the factor by which area in the coordinate domain (u, v) becomes scaled after mapped onto the tangent plane. It is the *jacobian* of the map between the plane (u, v) and the surface.

Suppose we have a mass distribution along our surface, given by $m(u, v)$ in (u, v) coordinates. The total mass of the surface would be

$$\int_{(u,v)\in D} m(u, v) \|x_u \times x_v\| du\, dv$$

a double integral which reminds us to take into account the area scaling factor named *jacobian*

Jacobians appear whenever we have a transformation between spaces of the same dimension and we want to compute measures (areas, volumes or lengths).

The arc length of the curve $\phi(t)$ can be computed as

$$\int \|\phi'(t)\| dt = \int \|x_u u'(t) + x_v v'(t)\|\, dt = \int \sqrt{(x_u \cdot x_u)u'^2 + 2(x_u \cdot x_v)u'v' + (x_v \cdot x_v)v'^2}\, dt$$

The subradical expression is the *first fundamental form* or the *metric tensor*

$$I(u, v) = Eu^2 + 2Fuv + Gv^2$$

where

$$\begin{pmatrix} E & F \\ F & G \end{pmatrix} = \begin{pmatrix} x_u \cdot x_u & x_u \cdot x_v \\ x_u \cdot x_v & x_v \cdot x_v \end{pmatrix}$$

With it, we may perform measurements in terms of surface coordinates (u, v). For instance, arc length

$$\int \sqrt{Eu'^2 + 2Fu'v' + Gv'^2}\, dt$$

or surface area

$$\int_D \sqrt{EG - F^2}\, du dv$$

Again, the radical is the jacobian of the mapping from (u, v) to the intrinsic coordinates of the surface.

7 Interpolation and approximation

These two problems arise too often in geometrical modelling and graphing.

7.1 Lagrange interpolation formula

We are given $n+1$ points in space $P_0, P_1, \ldots, P_n$ (usually depicted as functional values of a real function, see figure 12). We need a function p whose graph touches all the given points. This is an *interpolation problem*. The workhorse theorem for solving it is the standard *Lagrange interpolation formula*.

Theorem 1 *Let $n+1$ points in the XY plane $P_0 = (x_0, y_0), P_1 = (x_1, y_1), P_2 = (x_2, y_2), \ldots, P_n = (x_n, y_n)$ with different abscissae. Then, there is one and only one polynomial p of degree n satisfying*

$$p(x_i) = y_i \qquad \text{for } i = 0, 1, \ldots, n$$

The Lagrange interpolation polynomial can be computed in many ways, one of which is the explicit formula

$$p(x) = \sum_{i=0}^{n} y_k \frac{\omega_k(x)}{\omega_k(x_k)} \tag{22}$$

where the ω factors are n-th degree polynomials given by

$$\omega_k(x) = (x - x_0)(x - x_1) \cdots (x - x_n)/(x - x_k) \tag{23}$$

It is obvious that

$$\frac{\omega_k(x_i)}{\omega_k(x_k)} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}$$

because no two x's are the same. So this gives a solution to our problem.

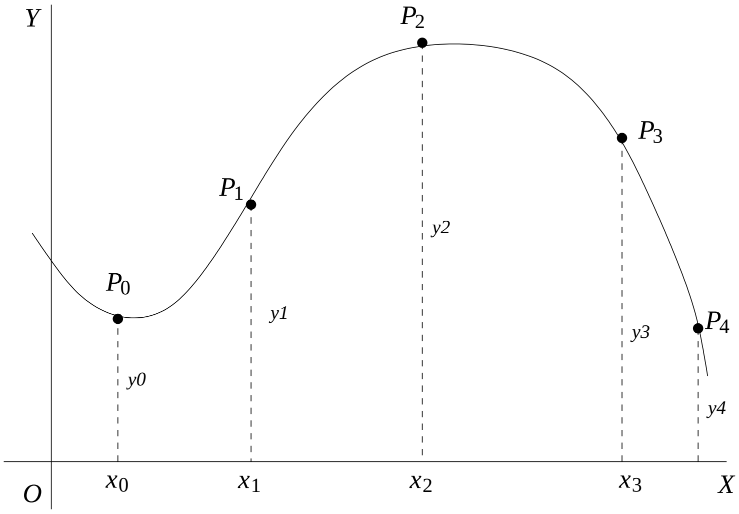
Formula (22) is not too useful for computation. Coefficients of the interpolating polynomial are better computed by solving a system of linear equations or by clever use of simmetries or the choice of the sample points (x_i, y_i).

An example: we want to find a quadratic polynomial interpolating the points $(-h, y_{-1})$, $(0, y_0)$ and $(h, y_1$. Instead of unrolling the nightmare of equation (22), recall that the interpolation process is *linear* and imagine what would happen if we had the sets of values

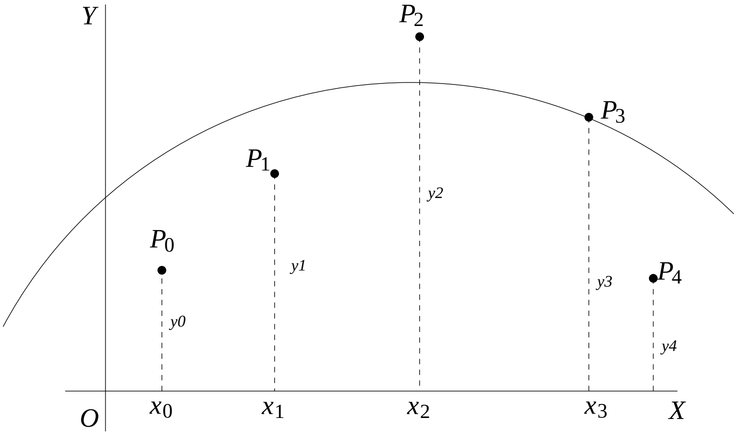
$$\begin{array}{lll} y_{-1} = 1 & y_0 = 0 & y_1 = 0 \\ y_{-1} = 0 & y_0 = 1 & y_1 = 0 \\ y_{-1} = 1 & y_0 = 0 & y_1 = 1 \end{array}$$

In the last case, we need a quadratic function with zeroes at $-h$ and 0, so it has the form

$$q_1(x) = Cx(x + h)$$


Y
P_2
P_3
P_1
y2
P_0
y1
y3
P_4
y0
y4
x_0
x_1
x_2
x_3
X
O

(a) Interpolation


Y
P_2
P_3
P_1
y2
P_0
y1
y3
P_4
y0
y4
x_0
x_1
x_2
x_3
X
O

(b) Approximation

Figure 12: Interpolation and approximation of point values

with C chosen to make $q_1(-1) = 1$, so that $C = 1/(1-h)$. By simmetry, the first function q_{-1} will be the mirror reflection of this one, that is $q_{-1} = 1/(1-h)\, x(h-x)$. And q_0 will be an even function by its simmetry, with zeroes in $-1, 1$, so it will be $(1-x^2)$ but for a constant factor, normalizing it to be 1 at the origin. Oh, the factor happens to be 1, so we get

$$q(x) = y_{-1}\frac{x(x+h)}{1-h} + y_0(1-x^2) + y_1\frac{x(h-x)}{1-h}$$

an easier way than going through (22).

7.2 Least squares fitting

A different problem is that of *approximation*: getting the curve to pass *near* the given points in a prescribed sense. This is an ample and difficult problem. The criteria of *nearness* are widely varied among applications, and we will see how this is approached (pun intended) by solution we describe in section 8.

We will restrict ourselves to the humblest, most primitive form of approximation technique, which is very useful however: *least squares fitting*. A common application of approximation techniques is solving an interpolation problem which is overdetermined, so that we have less unknowns than equations.

Let us be given, for instance, a set of points

$$P_i = (x_i, y_i)$$

with $i = 1 \dots n$, and let $n > 4$. Suppose you are forced to approximate a curve to them, but only a cubic polynomial. So you try to minimize

$$E(a,b,c,d) = \sum_{i=1}^{n}(y_i - ax_i^3 - bx_i^2 - cx_i - d)^2$$

This is at most a quadratic function of the parameters. To make it minimum, the partial derivatives of E should be zero:

$$\begin{aligned}
\frac{\partial E}{\partial a} &= -2\sum_{i=1}^{n} x_i^3(y_i - ax_i^3 - bx_i^2 - cx_i - d) &= 0\\
\frac{\partial E}{\partial b} &= -2\sum_{i=1}^{n} x_i^2(y_i - ax_i^3 - bx_i^2 - cx_i - d) &= 0\\
\frac{\partial E}{\partial c} &= -2\sum_{i=1}^{n} x_i(y_i - ax_i^3 - bx_i^2 - cx_i - d) &= 0\\
\frac{\partial E}{\partial d} &= -2\sum_{i=1}^{n} (y_i - ax_i^3 - bx_i^2 - cx_i - d) &= 0
\end{aligned}$$

Expanding and isolating the coefficients a, b, c, d we get a system

$$\begin{aligned}
\sum x_i^3 y_i &= a \sum x_i^6 + b \sum x_i^5 + c \sum x_i^4 + d \sum x_i^3 \\
\sum x_i^2 y_i &= a \sum x_i^5 + b \sum x_i^4 + c \sum x_i^3 + d \sum x_i^2 \\
\sum x_i^1 y_i &= a \sum x_i^4 + b \sum x_i^3 + c \sum x_i^2 + d \sum x_i^1 \\
\sum y_i &= a \sum x_i^3 + b \sum x_i + c \sum x_i^1 + dn
\end{aligned} \tag{24}$$

or, if you prefer

$$\begin{pmatrix} \sum x_i^6 & \sum x_i^5 & \sum x_i^4 & \sum x_i^3 \\ \sum x_i^5 & \sum x_i^4 & \sum x_i^3 & \sum x_i^2 \\ \sum x_i^4 & \sum x_i^3 & \sum x_i^2 & \sum x_i^1 \\ \sum x_i^3 & \sum x_i^2 & \sum x_i & n \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} \sum x_i^3 y_i \\ \sum x_i^2 y_i \\ \sum x_i^1 y_i \\ \sum y_i \end{pmatrix} \tag{25}$$

The coefficients a, b, c, d can be recovered from (25)

8 Modelling of curves and surfaces: splines

We are given $n + 1$ points $P_0, P_1, \ldots, P_n$, and we would like to construct a curve passing through these points, with the least possible contortion. This is a problem of *interpolation*. Some other times, the points given are merely *control points* and we wish the curve to pass *near* the points, or to pass through some of them without exactly touching the others. This is a problem of *approximation*.

In computer graphics, we usually solve this problem by means of pre-canned (pre-cooked?) functions with a definite set of parameters wich are to be computed to make the curve fit to our requirements. A set of standard solutions to this problems is at our disposal:

- Natural splines
- Hermite interpolants
- Bézier curves
- B-splines
- NURBS

8.1 Natural splines

A first approach is trying to interpolate by segments. The curve segment from P_i to P_{i+1} will be a function $p_i(t)$ with t ranging in the interval $(0, 1)$. The conditions will be four

$$\begin{aligned}
p_i(0) &= P_i \\
p_i(1) &= P_{i+1} \\
p_i'(1) &= p_{i+1}'(0) \\
p_i''(1) &= p_{i+1}''(0)
\end{aligned}$$

because each p_i is a cubic polynomial. Last two identities are regularity conditions, so that curvature does not have jumps.

The drawback of this approach is that each of the control points affect all the segments, forcing a recomputation if we change one single point. And solving each of the p_i coefficients implies solving a linear system of equations.

8.2 Hermite interpolants

A similar approach is Hermite interpolation, in which we prescribe the derivatives (tangents) at the ends of every interval:

$$\begin{aligned} p_i(0) &= P_i \\ p_i(1) &= P_{i+1} \\ p_i'(0) &= T_i \\ p_i'(1) &= T_{i+1} \end{aligned}$$

This requires specifying the tangents T_i at the control points, which is not always possible nor convenient. Regularity is lower than with natural splines; those are the prices of local control.

8.3 Bézier curves

This solution is based in the so-called *Bernstein polynomials*, which is a family of polynomials in the range $(0, 1)$ with nice properties of uniform approximation. Given a function $P(u)$ with $u \in (0, 1)$, its n-th Bernstein polynomial $B_n(P, u)$ is given by the formula

$$B_n(P, u) = \sum_{k=0}^{n} \binom{n}{k} P(k/n)\, t^k (1-t)^{n-k} \tag{26}$$

If we are given $n + 1$ control points, we draw the *Bézier curve* by using the parametric expression

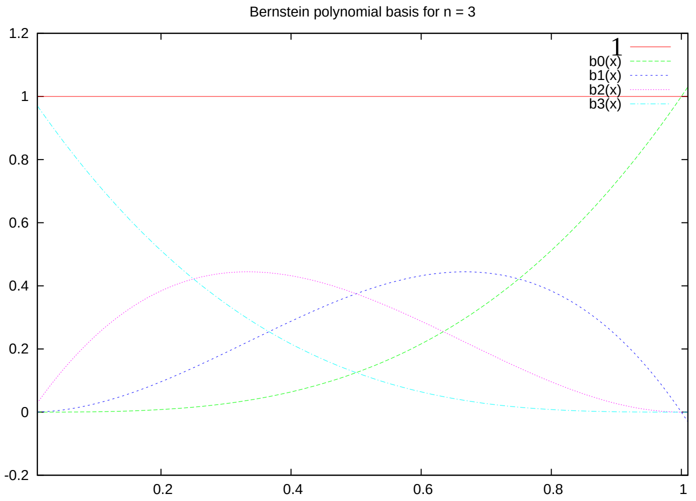
$$B_n(u) = \sum_{k=0}^{n} P_k \binom{n}{k} t^k (1-t)^{n-k} \qquad 0 \le u \le 1 \tag{27}$$

As usual in interpolation and approximation problems, the Bézier weight functions $\binom{n}{k} t^k (1-t)^{n-k}$ constitute a partition of unity

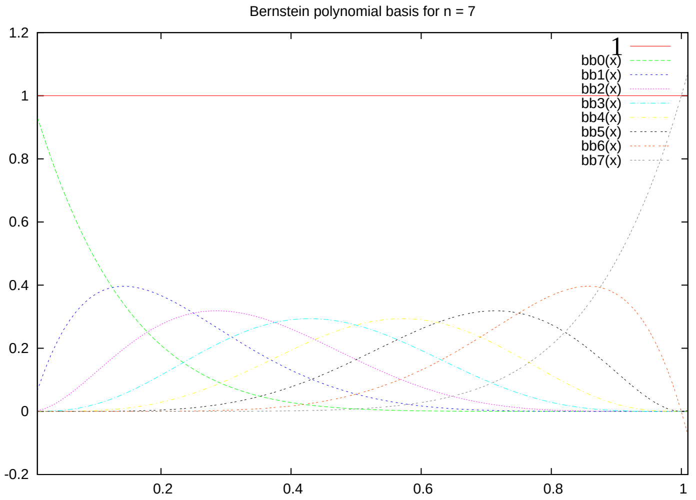
$$\sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} = 1$$

which, with positivity, makes the approximation process convex: the Bézier curve lies within the convex hull of the control points. Plots of the Bézier basis functions are shown in figure 13.

The most extended use of Bézier curves restricts them to cubic patches with four control points. It is an easy exercise (do it!) that P_0P_1 and P_2P_3 determine the tangent (the speed, in fact) at the ends, and the curve passes through the end control points P_0 and P_3, as seen in figure 14



(a) $n = 3$



(b) $n = 8$

Figure 13: Bézier basis for $n = 3$ and $n = 7$

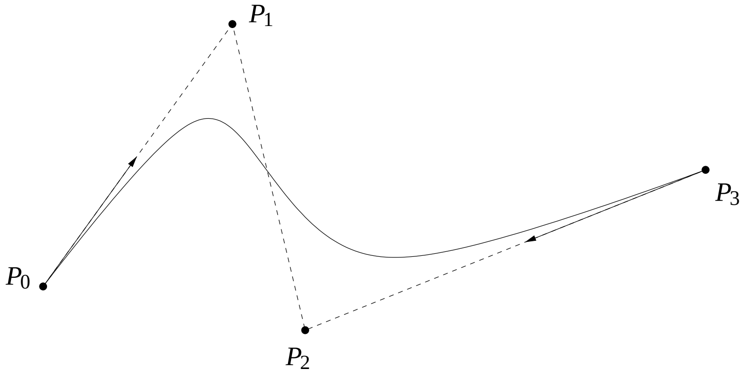


Figure 14: Bézier curve fitting end control points and with tangents pointing to middle control points

8.4 B-splines

A problem with standard interpolation techniques is that the whole curve depends on every control point; changing one of these implies recomputing (and redrawing) all the solution.

B-splines are a particular set of interpolants without this defficiency. Let us suppose we have $n+1$ control points $P_0, P_1, \ldots, P_n$. The solution curve will be

$$P(u) = \sum_{k=0}^{n} P_k B_{k,d}(u)$$

where the functions $B_{k,d}$ are d*-order* B-splines whose domain of definition and properties we are about to define.

The $B_{k,d}$ functions will be constructed on an interval $u_{\min}, u_{\max}$ divided in $n+d$ subintervals by points called *knots*, so that

$$\{u_{\min} = u_0, u_1, u_2, \ldots, u_k + d = u_{\max}\}$$

is the domain of the $B_{k,d}$. Each such function will be zero except in $(u_k, u_k + d)$ (spanning d subintervals of the range) where it will match a polynomial function of degree $d-1$ with positive values. We can see, then, that every value $P(u)$ will be influenced by d control points.

How are this magic functions constructed? By the *Cox-De Boor* recursion formula:

$$\begin{aligned} B_{k,1} &= \begin{cases} 1 & \text{if } u_k \leq u \\ 0 & \text{otherwise} \end{cases} \\ B_{k,d} &= \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1} + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1} \end{aligned} \tag{28}$$

Despite its impressive appearance, recurrence (28) is quite easy to apply. We can see an example of what results from it in figure 8.4. We have chosen a knot set

{0, 2, 3, 6, 7, 8, 11, 13} and constructed by hand (well, almost) the B-splines for that set until order four (i.e., cubic polynomials). From the plots, B-spline properties are quite apperent.

In subfigure 15e another interesting property appears: B-splines of any degree constitute a *partition of unity*, i.e., they are positive functions with unit sum. This is a crucial property making the interpolation process convex. In figure 8.4 we see the B-spline curve of degree three with control points

$$\begin{aligned} P_0 &= (0,0) & P_1 &= (1,1) \\ P_2 &= (0,2) & P_3 &= (1,3) \\ P_4 &= (0,4) \end{aligned}$$

This time, we have chosen uniform spacing for knots. We may appreciate how the curve is contained in the convex hull of the control points, which is a consequence of the positivity of the B-spline basis, and of its sum being equal to 1.

8.5 NURBS

B-splines can be applied with no changes if we work with projective coordinates, and we then obtain NURBS (Non-Uniform Rational B-Splines). The resulting formula

$$P(u) = \frac{\sum_{k=0}^{n} P_k w_k B_{k,d}(u)}{\sum_{k=0}^{n} w_k B_{k,d}(u)} \tag{29}$$

in which the denominator can be taken as the homogeneous component of the control points in projective coordinates, incorporates also weight factors w_k to the effect that the basis functions are now rational functions of the parameter.

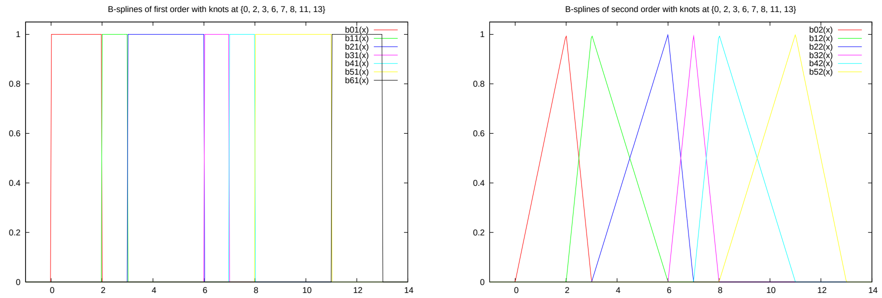
The main advantage of this is *projective invariance*: a non-rational B-spline, transformed by projection, need not be a B-spline anymore. The B-spline of the transformed control points is not the transformed of the B-spline original curve. This does hold for rational B-splines, so transforming them reduces to compute with control points only.

8.6 Splines for parametric surfaces

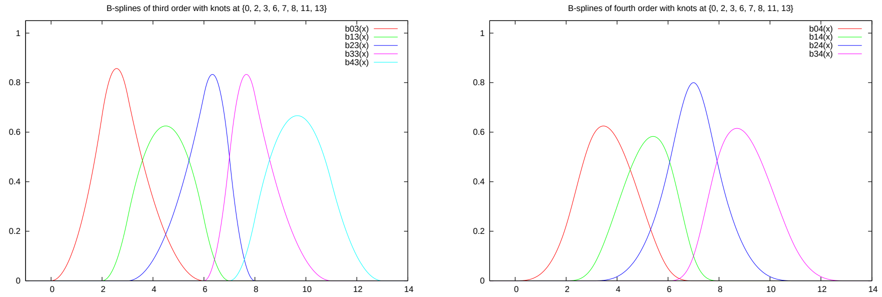
When dealing with surfaces, we may apply the same techniques as for curves verbatim. Now that we have two parameters, u, v, we need as a basis the tensor produc of spline bases in each of them. For instance, a surface patch can be interpolated through a net of control points P_{ij} by the formula

$$\sum_{i,j} P_{ij} B_{i,d}(u) B_{j,d}(v)$$

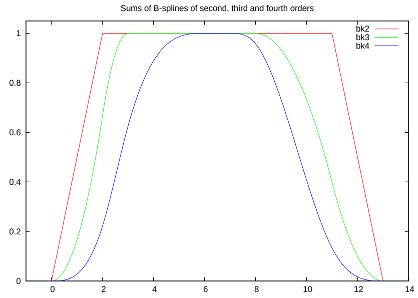
The partition of unity property holds for the product basis. The same trick applies to Bézier or NURBS patches.



(a) Order 1 (b) Order 2



(c) Order 3 (d) Order 4



(e) Sum of the above

Figure 15: Graphs of B-spline functions for a knot set {0, 2, 3, 6, 7, 8, 11, 13}. In 15e the partition of unity property is checked

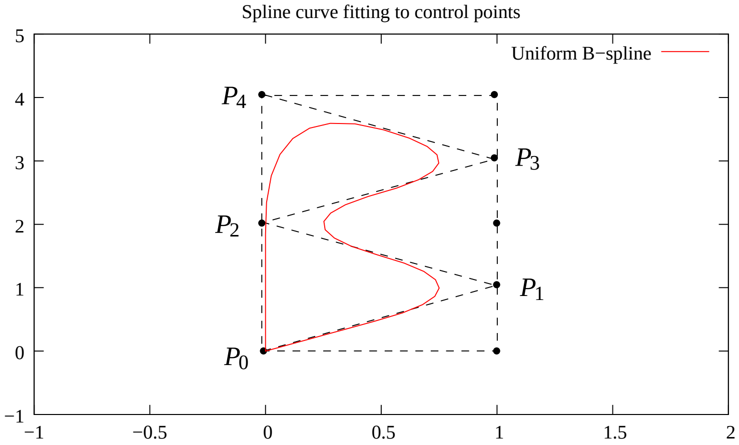


Figure 16: Uniform B-spline 3rd-degree approximation to five control points