Basics of 2D and 3D graphics

Transformations

Piotr Fulmański

piotr@fulmanski.pl

December 8, 2016

Spis treści

Let's start with translations. It seems to be the simplest one, but we faced with a problem: translation (destination of translation) $\mathbf{p}' = [p'_x \ p'_y \ p'_z]^T$ of a point $\mathbf{p} = [p_x \ p_y \ p_z]$ can be described by addition of two vectors: source $\mathbf{p} = [p_x \ p_y \ p_z]^T$ and translation (move) $\mathbf{t} = [t_x \ t_y \ t_z]^T$.

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \mathbf{t} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \end{bmatrix}$$

Because, as we will see, scaling and rotation are described by matrix and vector multiplication, this is a form we also want to have for translation.

The following matrix T translates a point $\mathbf{p} = [p_x \ p_y \ p_z]$ by the vector $\mathbf{t} = [t_x \ t_y \ t_z]$ ($\mathbf{p}'$ is the translated point)

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

In consequence we have

$$\mathbf{p}' = \mathbf{p} + \mathbf{t} \quad = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix}$$

As we can see, to have matrix form for translation we have to use concept of homogeneous coordinates.

The following matrix S scales the point $\mathbf{p} = [p_x \ \ p_y \ \ p_z]$ by a factor s_x along the X-axis, s_y along the Y-axis, and s_z along the Z-axis

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

In consequence we have

$$\mathbf{p}' = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \\ 1 \end{bmatrix}$$

The following matrix H sheares the point $\mathbf{p} = [p_x \; p_y \; p_z]$ by a factor s_{x1}, s_{x2} along the X-axis, s_{y1}, s_{y2} along the Y-axis, and s_{z1}, s_{z2} along the Z-axis

$$H = \begin{bmatrix} 1 & s_{x1} & s_2 & 0 \\ s_{y1} & 1 & s_{y2} & 0 \\ s_{z1} & s_{z2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In consequence we have

$$\mathbf{p}' = \begin{bmatrix} 1 & s_{x1} & s_2 & 0 \\ s_{y1} & 1 & s_{y2} & 0 \\ s_{z1} & s_{z2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + s_{x1}p_y + s_{x2}p_z \\ s_{y1}p_x + p_y + s_{y2}p_z \\ s_{z1}p_x + s_{z2}p_y + p_z \\ 1 \end{bmatrix}$$

The following matrix R reflects the point $\mathbf{p} = [p_x \;\; p_y \;\; p_z]$ through the xy plane

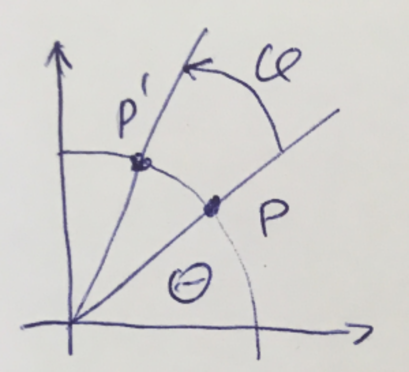
$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In consequence we have

$$\mathbf{p}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ -p_z \\ 1 \end{bmatrix}$$

Reflections through the xz and the yz planes are defined similarly.

For 2D case rotations are simple because of their uniqueness – there is only one axis to rotate about. Let's assume that we want to rotate point p by φ degrees to point p'.



If we want to rotate point p by φ degrees to point p', from *Basics of 2D and 3D graphics. Linear algebra* lecture, *Systems of coordinates. Polar coordinates* section, we know that

$$x = r \cos \varphi$$
$$y = r \sin \varphi$$

so we have simply that

$$p_x = |p| \cos(\theta)$$
$$p_y = |p| \sin(\theta)$$

and

$$p'_x = |p'| \cos(\theta + \varphi) \tag{1}$$
$$p'_y = |p'| \sin(\theta + \varphi) \tag{2}$$

Because we are dealing with rotations about the origin, thus we have

$$|p'| = |p|.$$

Using the trigonometric identities for the sum of angles we have that

$$p'_x = |p|\cos(\theta)\cos(\varphi) - |p|\sin(\theta)\sin(\varphi)$$
$$p'_y = |p|\sin(\theta)\cos(\varphi) + |p|\cos(\theta)\sin(\varphi)$$

and (1)-(2) we have that

$$\begin{aligned} p'_x &= p_x\cos(\varphi) - p_y\sin(\varphi) \\ p'_y &= p_y\cos(\varphi) + p_x\sin(\varphi) \end{aligned}$$

Pushing this into matrix form

$$R_{xy} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix}$$

In consequence we have

$$\mathbf{p}' = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} p_x\cos(\varphi) - p_y\sin(\varphi) \\ p_y\cos(\varphi) + p_x\sin(\varphi) \end{bmatrix}$$

In three dimensions the case is more complicated because the axis of rotation can be freely selected. The best case is if it is one of the Cartesian coordinates axis. In such a case the rotation matrices take the simple forms, which are a simple generalization of the 2D case. The easiest way to note this is whenthe rotation axis is the axis OZ. Then change the coordinate values apply only to the plane OXY and rotation matrix takes the form

$$R_{OZ} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By analogy we obtain rotation matrices when rotation axis is axis OX

$$R_{OX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and OY

$$R_{OY} = \begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The sequence of three rotations R_{OX}, R_{OY} and R_{OZ} can perform any rotation about an axis passing through the origin.
- The order of rotations matters.
- $R(-\varphi) = R^{-1}(\varphi) = R^T(\varphi)$.

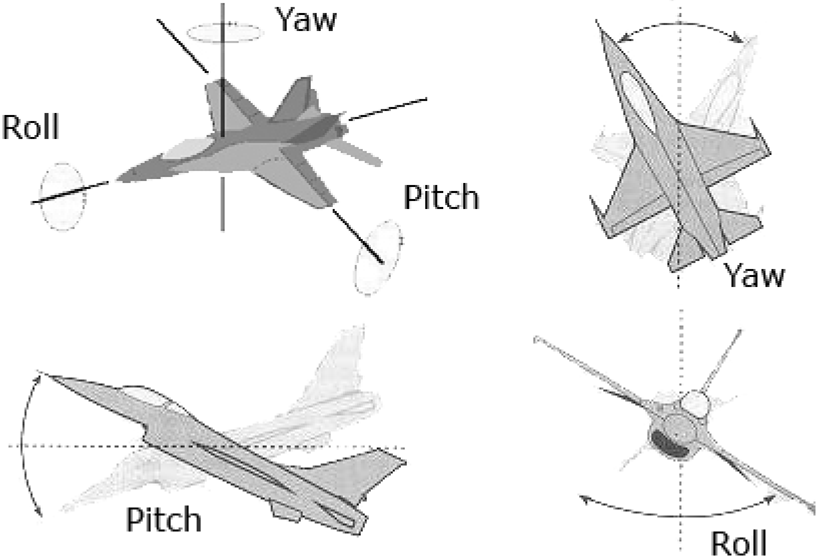
The Euler angles are three angles used to describe the orientation of a rigid body with respect to a fixed coordinate system. Any orientation can be achieved by composing three elemental rotations, i.e. rotations about the axes of a coordinate system. There exist twelve possible sequences of rotation axes, divided in two groups:

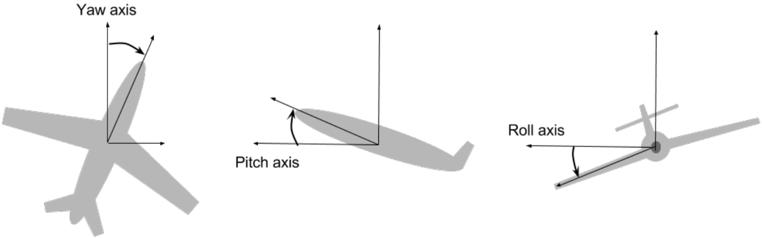
- proper Euler angles (z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y),
- Tait–Bryan angles (x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z).

Tait–Bryan angles are also called *Cardan angles* or *yaw-pitch-roll* (see next slides for explanation). Sometimes, both kinds of sequences are called Euler angles. In that case, the sequences of the first group are called proper or classic Euler angles.
Be aware that although Euler angles are typically denoted as α, β and γ, or φ, θ and ψ, different authors may use different sets of rotation axes to define Euler angles, or different names for the same angles.

Atomic transformation matrices

Rotations – 3D case: yaw-pitch-roll



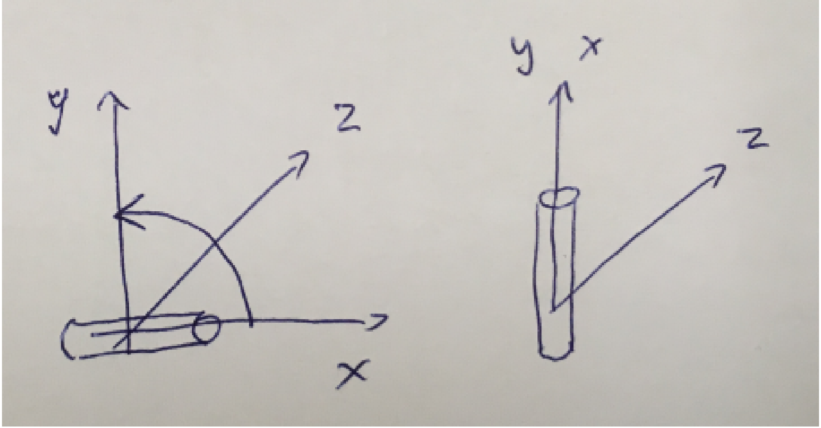


Yaw axis
Pitch axis
Roll axis

There are few problems using Eulers angles.

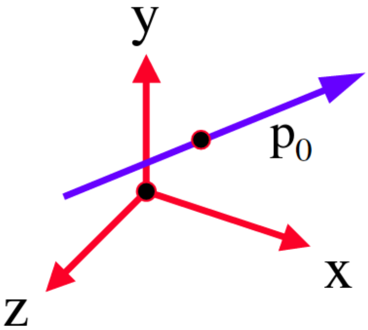
- There are many different Eulers angles resulting the same rotation.
- It is difficult to find Eulers angles to rotate an object in desired direction.
- In consequence it's difficult to compare two different rotatins or make interpolation of rotation.
- We can loss of one degree of freedom (so called *gimbal lock*). Loosing degree of freedom in this case means that one of the elemental rotation has no effect; we can make this rotation but it doesn't affect our object. For example, if an object is rotated 90° about the z-axis, the x-and y-axes will become one and the same.

Technically, gimbal lock is the loss of one degree of freedom in a three-dimensional, three-gimbal mechanism that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space.
The word lock is misleading: no gimbal is restrained. All three gimbals can still rotate freely about their respective axes of suspension. Nevertheless, because of the parallel orientation of two of the gimbals' axes there is no gimbal available to accommodate rotation along one axis.
For example, if an object is rotated 90° about the z-axis, the x-and y-axes will become one and the same.

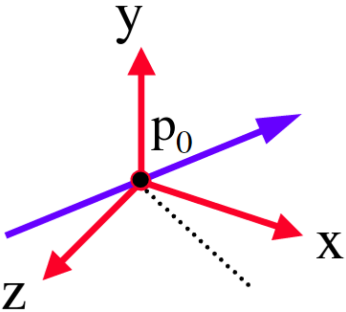

y
z
x
y
x
z

Assumption: axis of rotation can be located at any point p_0. Thus we have six degree of freedom.
The idea is simple: move the point to the origin, make the axis coincident with one of the coordinate axes, rotate, and then transform back.

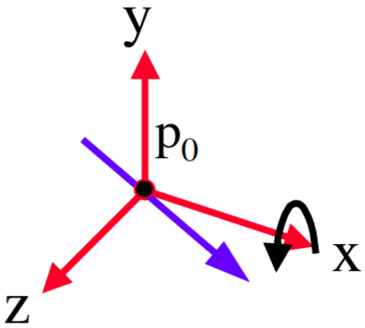
Assume that the axis passes through the point p_0.



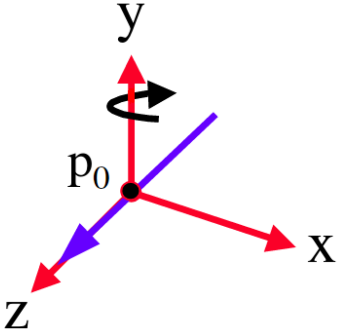
Translate p_0 to the origin.



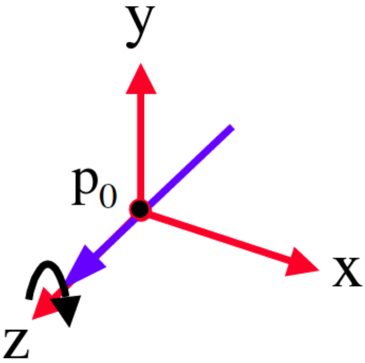
Rotate about the x-axis into the xz plane.



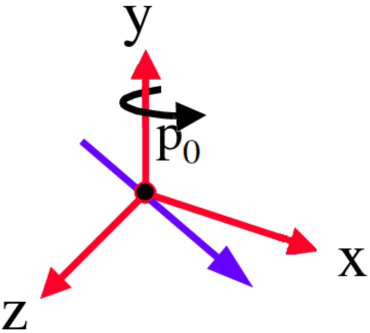
Rotate about the y-axis onto the z-axis.



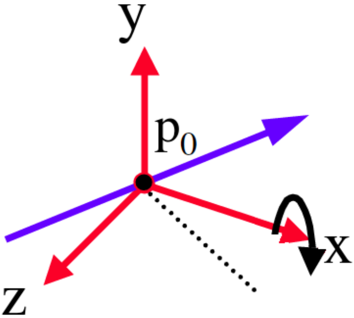
Rotate as needed about the z-axis.



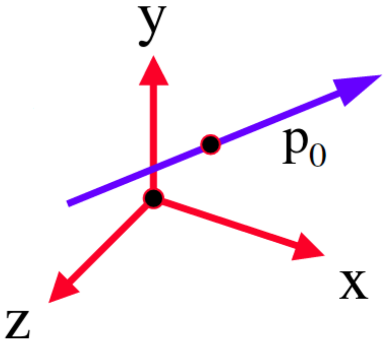
Apply inverse rotations about y.



Apply inverse rotations about x.



Apply inverse translation.



Given a unit vector $\mathbf{u} = (u_x, u_y, u_z)$, the matrix for a rotation by an angle of φ about an axis in the direction of $\mathbf{u}$ is defined as

$$R = \begin{bmatrix} c + u_x^2 (1-c) & u_x u_y (1-c) - u_z s & u_x u_z (1-c) + u_y s \\ u_y u_x (1-c) + u_z s & c + u_y^2 (1-c) & u_y u_z (1-c) - u_x s \\ u_z u_x (1-c) - u_y s & u_z u_y (1-c) + u_x s & c + u_z^2 (1-c) \end{bmatrix}$$

where $s = \sin\varphi$ and $c = \cos\varphi$.

The rotation representation we've been talking so far is known as *matrix representaion of a rotation*. Problems with this representation are

- We need to much floating-point values (nine while we just have three degrees of freedom).
- As a consequence of previous: expensive calculation.
- It's hard to find intermediate rotations between two known rotations.

We can think about quaternions like an extension to complex numbers. A number of the form

$$a + 0i + 0j + 0k,$$

where a is a real number, is called **real**, and a number of the form

$$0 + bi + cj + dk,$$

where b, c, and d are real numbers, is called **pure imaginary**. If

$$a + bi + cj + dk$$

is any quaternion, then a is called its **scalar part** and $bi + cj + dk$ is called its **vector part**. The scalar part of a quaternion is always real, and the vector part is always pure imaginary. Even though every quaternion is a vector in a four-dimensional vector space, it is common to define a vector to mean a pure imaginary quaternion. With this convention, a vector is the same as an element of the vector space R^3.
Hamilton called pure imaginary quaternions **right quaternions** and real numbers (considered as quaternions with zero vector part) **scalar quaternions**.

```
struct QUATERNION
{
  float x, y, z, w;

  QUATERNION() { }
  QUATERNION(float x, float y, float z, float w):
    x(x), y(y), z(z), w(w) { }
};
```

Jednostkowy kwaternion można utożsamiać z obrotem w przestrzeni 3D. Kwaternion tworzy sie podajac jednostkowy wektor, którego kierunek wskazuje oś obrotu oraz kat, o jaki chcemy obracać wokół tego wektora (zwykle w radianach).
Informacji tych nie wpisujemy jednak do składowych kwaterniona bezpośrednio. Trzeba je zakodować według algorytmu jak na poniższym listingu, obliczajac najpierw sinus i cosinus połowy podanego kata.

```
void AxisToQuaternion(QUATERNION *Out,
                      const VEC3 &Axis,
                      float Angle)
{
  Angle *= 0.5f;
  float Sin = sinf(Angle);
  Out->x = Sin * Axis.x;
  Out->y = Sin * Axis.y;
  Out->z = Sin * Axis.z;
  Out->w = cosf(Angle);
}
```

Przypadkiem szczególnym jest obracanie wokół osi X, Y lub Z. Algorytm znacznie sie wówczas upraszcza i dla optymalizacji warto przygotować osobne funkcje. Poniżej funkcja dla obrotu wokół osi X; dla pozostałych przypadków należy postapić analogicznie.

```
void QuaternionRotationX(QUATERNION *Out, float a)
{
  a *= 0.5f;
  Out->x = sinf(a);
  Out->y = 0.0f;
  Out->z = 0.0f;
  Out->w = cosf(a);
}
```

As a set, the quaternions $\mathbb{H}$ are equal to R^4, a four-dimensional vector space over the real numbers. The quaternions looks a lot like a four-dimensional vector, but it behaves quite differently.
$\mathbb{H}$ has three operations: addition, scalar multiplication, and quaternion multiplication.

Quaternions support some of the familiar operations from vector algebra, such as vector addition. We have see a formula for addition – to remember it, if

$$q = (r, v),\ \ q \in \mathbb{H},\ \ r \in R,\ \ v \in R^3$$

then

$$(r_1,\ v_1) + (r_2,\ v_2) = (r_1 + r_2,\ v_1 + v_2).$$

However, we must remember that **the sum of two unit quaternions does not represent a 3D rotation, because such a quaternion would not be of unit length**.

One of the most important operations we will perform on quaternions is that of multiplication. Given two quaternions p and q representing two rotations P and Q, respectively, the product pq represents the composite rotation (i.e., rotation Q followed by rotation P[1]). We will restrict to the multiplication which is used in conjunction with 3D rotations, namely the Grassman product. If

$$q = (r, v), \;\; q \in \mathbb{H}, \;\; r \in R, \;\; v \in R^3$$

then

$$(r_1, v_1)(r_2, v_2) = (r_1 r_2 - v_1 \cdot v_2, r_1 v_2 + r_2 v_1 + v_1 \times v_2).$$

[1] **Mind the order!**

If

$$q = (r, v),\ \ q \in \mathbb{H},\ \ r \in R,\ \ v \in R^3$$

then norm $|q|$ is defined as follows

$$|q| = \sqrt{q\overline{q}} = \sqrt{\overline{q}q} = \sqrt{r^2 + v_x^2 + v_y^2 + v_z^2},$$

where $\overline{q}$ denotes conjugation (to be explain). To normalize vector the following formula have to be used

$$\text{normalize}(q) = \frac{q}{|q|} = \left[\frac{v_x}{|q|}\ \frac{v_y}{|q|}\ \frac{v_z}{|q|}\ \frac{r}{|q|} \right].$$

Conjugate of a quaternion q is defined as follows

$$\overline{q} = (r, -v)$$

where

$$q = (r, v), \; q \in \mathbb{H}, \; r \in R, \; v \in R^3.$$

The inverse of a quaternion q is denoted q^{-1} and is defined as a quaternion which, when multiplied by the original, yields the scalar 1 (i.e., $qq^{-1} = 0i + 0j + 0k + 1$)

$$q^{-1} = \frac{\overline{q}}{|q|^2}$$

where

$$q = (r, v), \;\; q \in \mathbb{H}, \;\; r \in R, \;\; v \in R^3.$$

What is nice, because in computer games quaternions represent 3D rotations, they are always of unit length. So, for our purposes, the inverse and the conjugate are identical:

$$q^{-1} = \overline{q}$$

where

$$q = (r, v),\ \ q \in \mathbb{H},\ \ r \in R,\ \ v \in R^3.$$

Other properties

$$\overline{(pq)} = \overline{q}\,\overline{p},$$

$$(pq)^{-1} = q^{-1}p^{-1}.$$

Rewrite vector v in quaternion form v_q

$$v_q = (0, v) = [v_x \; v_y \; v_z \; 0].$$

The rotated vector v' by a quaternion q can be found as follows

$$v' = \text{rotate}(v, q) = q v_q q^{-1}.$$

Consider three distinct rotations, represented by the quaternions q_1, q_2 and q_3. We want to apply rotation 1 first, followed by rotation 2 and finally rotation3. The composite rotation quaternion q_{comp} can be found and applied to vector v (in its quaternion form, v_q) to get rotated vector v' as follows

$$v' = q_3 q_2 q_1 v_q q_1^{-1} q_2^{-1} q_3^{-1} = q_{comp} v_q q_{comp}^{-1}.$$

If we let

$$q = (r, v) = [v_x \ v_y \ v_z \ r] = [x \ y \ z \ w]$$

then matrix representation of 3D rotation M we can find as follow

$$M = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2zw & 2xz - 2yw \\ 2xy - 2zw & 1 - 2x^2 - 2z^2 & 2yz + 2xw \\ 2xz + 2yw & 2yz - 2xw & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

Given two quaternions p and q representing rotations A and B, we can find an intermediate rotation q_{LERP} that is t percent of the way from A to B as follows

$$
\begin{aligned}
q_{\text{LERP}} &= \text{LERP}(p,q,t) = \tfrac{(1-t)p+tq}{|(1-t)p+tq|} \\
&= \text{normalize}\left(\begin{bmatrix} (1-t)p_x + tq_x \\ (1-t)p_y + tq_y \\ (1-t)p_z + tq_z \\ (1-t)p_r + tq_r \end{bmatrix}^T \right).
\end{aligned}
$$

The problem with the LERP is that it effectively interpolates along a chord of the hypersphere, rather than along the surface of the hypersphere itself. This leads to rotational animations that do not have a constant angular speed when the parameter t is changing at a constant rate. The rotation will appear slower at the end points and faster in the middle of the animation.
To solve this problem, we can use a variant of the LERP operation known as spherical linear interpolation, or SLERP for short

$$\mathrm{SLERP}(p, q, t) = t_p p + t_q q,$$

where

$$\begin{aligned} t_p &= \tfrac{\sin((1-t)\theta)}{\sin(\theta)}, \\ t_q &= \tfrac{\sin(t\theta)}{\sin(\theta)}, \end{aligned}$$

and

$$\theta = \arccos(p \cdot q).$$

Atomic transformation matrices

General form

From the preceding material we can conclude that all matrix transformations can be described on generalized 4×4 transformation matrix of the form

$$\begin{bmatrix} L & L & L & T \\ L & L & L & T \\ L & L & L & T \\ P & P & P & O \end{bmatrix}$$

where

- L – linear transformations (scaling, shearing, rotation, reflection),
- T – translation,
- O – overall scaling,
- P – perspective transformation.

We have the following problem: given independent vectors u and v and any two vectors x and y , find a linear transformation, in matrix form, that sends u to x and v to y.

Let M be the matrix whose columns are u and v. Then

$$T : x \to Mx$$

sends e_x to u and e_y to v.
Therefore

$$T^{-1} : x \to M^{-1}x$$

sends u to e_x and v to e_y.

Let K be the matrix whose columns are x and y. Then

$$R : x \to Kx$$

sends e_x to x and e_y to y.

Applying first T^{-1} and then R to vector u we send it to x (via e_1). Smilarly for v.

$$R(T^{-1}) : x \to KM^{-1}x$$

Thus, the matrix for the transformation sending the vectors u to the x and v to the y is just KM^{-1}.

In the special case when we want to go from the usual coordinates (parent space) on a vector to its coordinates in some coordinate system (child space) with basis vectors u, v, which are

- unit vectors
- and mutually perpendicular,
- vectors u and v are expressed in parent space

the transformation matrix is one whose **rows** are the transposes of u and v

For example, if

$$u = \begin{bmatrix} \frac{3}{5} \\ \frac{4}{5} \end{bmatrix}$$

and

$$v = \begin{bmatrix} -\frac{4}{5} \\ \frac{3}{5} \end{bmatrix},$$

then the point p_P from the xy parent space

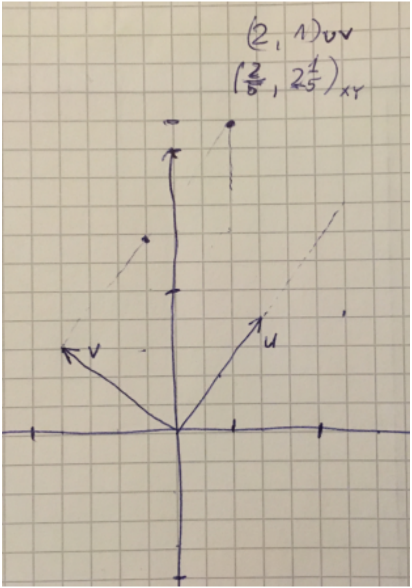
$$p_P = \begin{bmatrix} \frac{2}{5} \\ \frac{11}{5} \end{bmatrix},$$

expressed in uv child coordinates, is

$$p_C = \begin{bmatrix} \frac{3}{5} & \frac{4}{5} \\ -\frac{4}{5} & \frac{3}{5} \end{bmatrix} \begin{bmatrix} \frac{2}{5} \\ \frac{11}{5} \end{bmatrix} = \begin{bmatrix} \frac{6}{25} + \frac{44}{25} \\ -\frac{8}{25} + \frac{33}{25} \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Coordinate system transformation

Transform from parent space to child space



Coordinate system transformation

Transform from child space to parent space

Conversely, when we want to go from some coordinate system (child space) on a vector to its coordinates in the usual coordinates (parent space) with basis vectors u, v, the transformation matrix is one whose **columns** are u and v

For example, if

$$u = \begin{bmatrix} \frac{3}{5} \\ \frac{4}{5} \end{bmatrix}$$

and

$$v = \begin{bmatrix} -\frac{4}{5} \\ \frac{3}{5} \end{bmatrix},$$

then the point p_C from the uv child space

$$p_C = \begin{bmatrix} 2 \\ 1 \end{bmatrix},$$

expressed in xy prent coordinates, is

$$p_P = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{6}{5} - \frac{4}{5} \\ \frac{8}{5} + \frac{3}{5} \end{bmatrix} = \begin{bmatrix} \frac{2}{5} \\ \frac{11}{5} \end{bmatrix}.$$

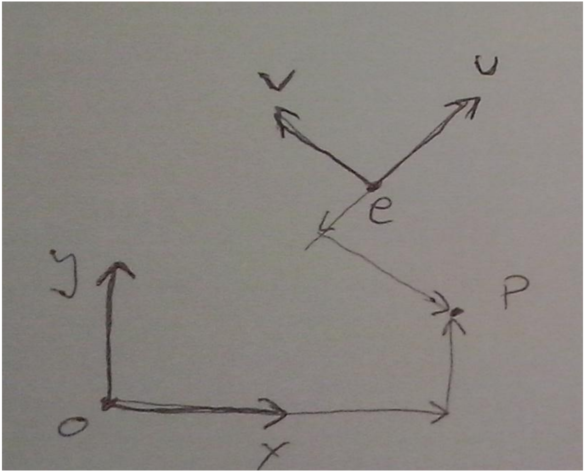
From geometric point of view a coordinate system (or coordinate frame) consists of an origin and a basis which is a set of vectors. A basis in most cases is orthonormal (which means that vectors are orthonormal, that is, they are all unit vectors and orthogonal to each other).
In 2D case with origin $\mathbf{e}$ and basis $\{\mathbf{u}, \mathbf{v}\}$, the coordinates (p_u, p_v) describe the point

$$\mathbf{p} = \mathbf{e} + p_u \mathbf{u} + p_v \mathbf{v}.$$

Similarly, we can express point $\mathbf{p}$ in terms of another coordinate system

$$\mathbf{p} = \mathbf{o} + p_x \mathbf{x} + p_y \mathbf{y}$$

(see next slide).



v
u
e
y
P
o
x

We can express this relationship using matrix transformation

$$\begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & e_x \\ u_y & v_y & e_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_u \\ p_v \\ 1 \end{bmatrix}$$

Note that this assumes we have the point **e** and vectors **u** and **v** stored in some canonical coordinates which is in this case from the (x, y)-coordinate system.

In most cases we write this matrix like this

$$p_{xy} = \begin{bmatrix} u & v & e \\ 0 & 0 & 1 \end{bmatrix} p_{uv}$$

It takes points expressed in the (u, v) coordinate system and converts them to the same points expressed in the (x, y) coordinate system (but (u, v) coordinate system – a child system – has to be described in the (x, y) coordinate system – parent system – terms).

Consider a following example:

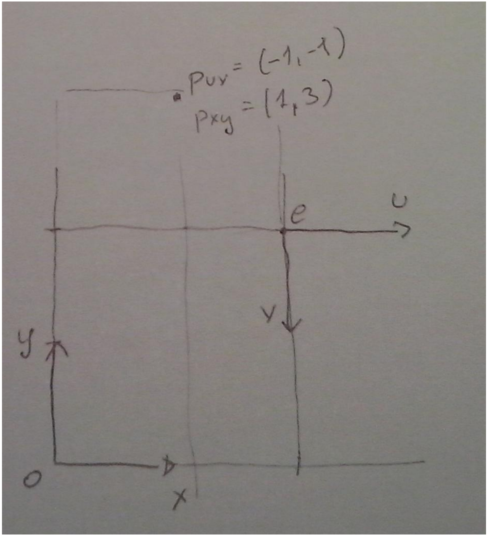
- $e = (2, 2)$
- $u = (1, 0)$
- $v = (0, -1)$
- $p_{uv} = (-1, -1)$

so

$$p_{xy} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} p_{uv}$$

and finally

$$p_{xy} = (1, 3, 1)$$



$p_{uv} = (-1,-1)$
$p_{xy} = (1,3)$
e
u
v
y
o
x

In 3D case we have

$$p_{xyz} = \begin{bmatrix} u & v & w & e \\ 0 & 0 & 0 & 1 \end{bmatrix} p_{uvw}$$

Any child-space position vector p_C can be transformed into a parent-space position vector p_P as follows

$$p_P = M_{C \to P} p_C$$

where transformation matrix

$$M_{C \to P} = \left[\begin{array}{cccc} \mathbf{u}_C & \mathbf{v}_C & \mathbf{w}_C & \mathbf{t}_C \end{array} \right]$$

and

- $\mathbf{u}_C$ is the unit basis vector along the child space X-axis, expressed **in parent space coordinates**;
- $\mathbf{v}_C$ is the unit basis vector along the child space Y-axis, **in parent space**;
- $\mathbf{w}_C$ is the unit basis vector along the child space Z-axis, **in parent space**;
- $\mathbf{t}_C$ is the translation of the child coordinates system relative **to parent space**.

We can calculate orthonormal basis that is aligned with a given vector.
That is, given a vector $\mathbf{a}$, we want an orthonormal $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ such that $\mathbf{w}$ points in the same direction as $\mathbf{a}$.
This can be done using cross products as follows.
First make $\mathbf{w}$ a unit vector in the direction of $\mathbf{a}$:

$$\mathbf{w} = \frac{\mathbf{a}}{||\mathbf{a}||}$$

Then choose any vector $\mathbf{t}$ not collinear with $\mathbf{w}$, and use the cross product to build a unit vector $\mathbf{u}$ perpendicular to $\mathbf{w}$:

$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{||\mathbf{t} \times \mathbf{w}||}.$$

Once $\mathbf{w}$ and $\mathbf{u}$ are in hand, completing the basis is simple:

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$