9 Rendering Gems

9.1 Introduction

The previous chapters have considered the rendering pipeline in a logical order from start to finish. However, here we present a number of tips and tricks which failed to fit neatly into any category. Some of the gems are well known, and are simply presented here for completeness, while others are original.

9.2 Useful Matrices

Matrices are used throughout the rendering system, performing transformations such as translation, scaling and rotation. Here we collect together various matrices which are used to represent the most common transformations. The matrices are given in row-major form and assume that points are represented as row vectors and post-multiplied by the transformation matrix. That is

$$(x\ y\ z\ 1) * \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} = \begin{matrix} ax+ey+iz+m & bx+fy+jz+n & cx+gy+kz+o \\ +dx+hy+lz+p \end{matrix}$$

The matrices should be transposed if they are used to pre-multiply column vectors. Translation by `tx ty tz`:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{pmatrix}$$

Scaling by `sx sy sz`:

$$\begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation by angle `theta` about axis `ax ay az`: The axis vector should be normalized.

```
s = sin(theta)
c = cos(theta)
t = 1-c
⎛ ax*ax*t+c     ay*ax*t+s*az  az*ax*t-s*ay  0 ⎞
⎜ ax*ay*t-s*az  ay*ay*t+c     az*ay*t+s*ax  0 ⎟
⎜ ax*az*t+s*ay  ay*az*t-s*ax  az*az*t+c     0 ⎟
⎝ 0             0             0             1 ⎠
```

Skewing a point by angle `theta` from axis `dx dy dz` in the direction of axis `ex ey ez`: Both vectors should be normalized, and `theta` should be less than the angle between the two vectors.

```
maxangle = arccos(dx*ex+dy*ey+dz*ez)
s = sin(theta)/sin(maxangle-theta)
⎛ 1+s*dx*ex   s*dx*ey     s*dx*ez    0 ⎞
⎜   s*dy*ex 1+s*dy*ey     s*dy*ez    0 ⎟
⎜   s*dz*ex   s*dz*ey   1+s*dz*ez    0 ⎟
⎝    0          0           0       1 ⎠
```

Skew is little used and its implementation has been found to vary. The results of skewing in different rendering systems may not be consistent.

A perspective transformation given a field of view `fov` and `near` and `far` clipping planes: After the transformation, points inside the viewing angle have x and y coordinates between –1 and 1. Depth values are scaled so that points on the near clipping plane are at $z = 0$ and points on the far clipping plane lie at $z = 1$.

```
itan2 = 1/tan(fov/2)
⎛ itan2    0              0              0 ⎞
⎜   0    itan2            0              0 ⎟
⎜   0      0        far/(far-near)       1 ⎟
⎝   0      0    near*far/(near-far)      0 ⎠
```

9.3 Orientation and Handedness

One of the hardest things to get correct when developing a renderer which adheres to the RenderMan standard relates to the section titled "Orientation and Sides". While it is confusing for the renderer designer, the way RenderMan handles orientation does make life easy for the users of compliant renderers. By default, it guarantees that the outside surface will be visible. If the user wants to see the inside instead, a well-placed call to `RiReverseOrientation()` will usually suffice.