```
pangolin::OpenGlRenderState s_cam(
        pangolin::ProjectionMatrix(1000, 600, 420, 420, 500, 300, 0.1, 1000),
        pangolin::ModelViewLookAt(3, 3, 3, 0, 0, 0, pangolin::AxisY)
    );
/// Object representing OpenGl Matrix.
struct PANGOLIN_EXPORT OpenGlMatrix {
        static OpenGlMatrix Translate(GLprecision x, GLprecision y, GLprecision z);
        static OpenGlMatrix Scale(GLprecision x, GLprecision y, GLprecision z);
        static OpenGlMatrix RotateX(GLprecision theta_rad);
        static OpenGlMatrix RotateY(GLprecision theta_rad);
        static OpenGlMatrix RotateZ(GLprecision theta_rad);



        template<typename P>
        static OpenGlMatrix ColMajor4x4(const P* col_major_4x4);

        OpenGlMatrix();

#ifdef USE_EIGEN
        template<typename P>
        OpenGlMatrix(const Eigen::Matrix<P,4,4>& mat);

        template<typename P>
        OpenGlMatrix(const Eigen::Transform<P,3,Eigen::Affine>& mat) : OpenGlMatrix(mat.matrix()) { }

        template<typename P>
        operator Eigen::Matrix<P,4,4>() const;

        template<typename P>
        operator Eigen::Transform<P,3,Eigen::Affine>() const;
#endif // USE_EIGEN

#ifdef HAVE_TOON
        OpenGlMatrix(const TooN::SE3<>& T);
        OpenGlMatrix(const TooN::Matrix<4,4>& M);
        operator const TooN::SE3<>() const;
        operator const TooN::Matrix<4,4>() const;
#endif // HAVE_TOON

#ifdef HAVE_OCULUS
        OpenGlMatrix(const OVR::Matrix4f& M);
        operator const OVR::Matrix4f() const;
#endif // HAVE_OCULUS

        // Load matrix on to OpenGl stack
        void Load() const;

        void Multiply() const;

        void SetIdentity();

        OpenGlMatrix Transpose() const;

        OpenGlMatrix Inverse() const;

        GLprecision& operator()(int r, int c) {
            return m[4*c +r];
        }

        GLprecision operator()(int r, int c) const {
            return m[4 * c + r];
        }

        // Column major Internal buffer
        GLprecision m[16];
};
/// Object representing attached OpenGl Matrices / transforms.
class PANGOLIN_EXPORT OpenGlRenderState
{
public:
        OpenGlRenderState();
        OpenGlRenderState(const OpenGlMatrix& projection_matrix);
        OpenGlRenderState(const    OpenGlMatrix&    projection_matrix,    const    OpenGlMatrix&
modelview_matrix);

        static void ApplyIdentity();

        void Apply() const;
        OpenGlRenderState& SetProjectionMatrix(OpenGlMatrix m);
        OpenGlRenderState& SetModelViewMatrix(OpenGlMatrix m);

        OpenGlMatrix& GetProjectionMatrix();
        OpenGlMatrix GetProjectionMatrix() const;

        OpenGlMatrix& GetModelViewMatrix();
        OpenGlMatrix GetModelViewMatrix() const;

        OpenGlMatrix GetProjectionModelViewMatrix() const;
        OpenGlMatrix GetProjectiveTextureMatrix() const;

        void EnableProjectiveTexturing() const;
        void DisableProjectiveTexturing() const;

        //! Seemlessly move OpenGl camera relative to changes in T_wc,
        //! whilst still enabling interaction
        void Follow(const OpenGlMatrix& T_wc, bool follow = true);
        void Unfollow();

        // Experimental - subject to change
        OpenGlMatrix& GetProjectionMatrix(unsigned int view);
        OpenGlMatrix GetProjectionMatrix(unsigned int view) const;
        OpenGlMatrix& GetViewOffset(unsigned int view);
        OpenGlMatrix GetViewOffset(unsigned int view) const;
        OpenGlMatrix GetModelViewMatrix(int i) const;
        void ApplyNView(int view) const;

        PANGOLIN_DEPRECATED("Use SetProjectionMatrix() or SetModelViewMatrix() instead.")
        OpenGlRenderState& Set(OpenGlMatrixSpec spec);

protected:
        OpenGlMatrix modelview;
        std::vector<OpenGlMatrix> projection;
        std::vector<OpenGlMatrix> modelview_premult;
        OpenGlMatrix T_cw;
        bool follow;
};

OpenGlRenderState::OpenGlRenderState(const OpenGlMatrix& projection_matrix, const
OpenGlMatrix& modelview_matrx)
        : modelview(modelview_matrx), follow(false)
{
        projection.push_back( projection_matrix );
}
```

```
// Use OpenGl's default frame of reference
OpenGlMatrixSpec ProjectionMatrix(int w, int h, GLprecision fu, GLprecision fv, GLprecision u0,
GLprecision v0, GLprecision zNear, GLprecision zFar )
{
	return ProjectionMatrixRUB_BottomLeft(w,h,fu,fv,u0,v0,zNear,zFar);
}
// Camera Axis:
//      X - Right, Y - Up, Z - Back
// Image Origin:
//      Bottom Left
// Caution: Principal point defined with respect to image origin (0,0) at
//                  bottom left of bottom-left pixel (not center, and in different frame
//                  of reference to projection function image)
OpenGlMatrixSpec ProjectionMatrixRUB_BottomLeft(int w, int h, GLprecision fu, GLprecision fv,
GLprecision u0, GLprecision v0, GLprecision zNear, GLprecision zFar )
{
	// http://www.songho.ca/opengl/gl_projectionmatrix.html
	const GLprecision L = +(u0) * zNear / -fu;
	const GLprecision T = +(v0) * zNear / fv;
	const GLprecision R = -(w-u0) * zNear / -fu;
	const GLprecision B = -(h-v0) * zNear / fv;

	OpenGlMatrixSpec P;
	P.type = GlProjectionStack;
	std::fill_n(P.m,4*4,0);

	P.m[0*4+0] = 2 * zNear / (R-L);
	P.m[1*4+1] = 2 * zNear / (T-B);
	P.m[2*4+2] = -(zFar +zNear) / (zFar - zNear);
	P.m[2*4+0] = (R+L)/(R-L);
	P.m[2*4+1] = (T+B)/(T-B);
	P.m[2*4+3] = -1.0;
	P.m[3*4+2] =    -(2*zFar*zNear)/(zFar-zNear);

	return P;
}
OpenGlMatrix ModelViewLookAt(GLprecision ex, GLprecision ey, GLprecision ez, GLprecision lx,
GLprecision ly, GLprecision lz, AxisDirection up)
{
	const GLprecision* u = AxisDirectionVector[up];
	return ModelViewLookAtRUB(ex,ey,ez,lx,ly,lz,u[0],u[1],u[2]);
```

```
}
/// Direction vector for each AxisDirection enum
const static GLprecision AxisDirectionVector[7][3] = {
	{0,0,0},
	{-1,0,0}, {1,0,0},
	{0,-1,0}, {0,1,0},
	{0,0,-1}, {0,0,1}
};
OpenGlMatrix ModelViewLookAtRUB(GLprecision ex, GLprecision ey, GLprecision ez, GLprecision lx,
GLprecision ly, GLprecision lz, GLprecision ux, GLprecision uy, GLprecision uz)
{
	OpenGlMatrix mat;
	GLprecision* m = mat.m;

	const GLprecision u_o[3] = {ux,uy,uz};

	GLprecision x[3], y[3];
	GLprecision z[] = {ex - lx, ey - ly, ez - lz};
	Normalise<3>(z);

	CrossProduct(x,u_o,z);
	CrossProduct(y,z,x);

	// Normalize x, y
	const GLprecision lenx = Length<3>(x);
	const GLprecision leny = Length<3>(y);

	if( lenx > 0 && leny > 0) {
		for(size_t r = 0; r < 3; ++r ) {
			x[r] /= lenx;
			y[r] /= leny;
		}
	#define M(row,col)	m[col*4+row]
		M(0,0) = x[0];
		M(0,1) = x[1];
		M(0,2) = x[2];
		M(1,0) = y[0];
		M(1,1) = y[1];
		M(1,2) = y[2];
		M(2,0) = z[0];
		M(2,1) = z[1];
```

```
        M(2,2) = z[2];
        M(3,0) = 0.0;
        M(3,1) = 0.0;
        M(3,2) = 0.0;
        M(0,3) = -(M(0,0)*ex + M(0,1)*ey + M(0,2)*ez);
        M(1,3) = -(M(1,0)*ex + M(1,1)*ey + M(1,2)*ez);
        M(2,3) = -(M(2,0)*ex + M(2,1)*ey + M(2,2)*ez);
        M(3,3) = 1.0;
    #undef M
        return mat;
    }else{
        throw std::invalid_argument("'Look' and 'up' vectors cannot be parallel when calling
ModelViewLookAt.");
    }

}
```