```
#include "glutwindow.hpp"
#include "ppmwriter.hpp"
#include "pixel.hpp"

#include <iostream>
#include <cmath>

#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

#include "vector.hpp"
#include "matrix.hpp"

#ifdef __APPLE__
  #include <GLUT/glut.h>
#else
  #include <GL/glut.h>
#endif

// this is a dummy raytrace application
class application
{
public :

  // raytracing is initiated from here
  void start()
  {
    // the following code might also be executed in any method
    // just start your raytracing algorithm from here

    // size of a tile in checkerboard
    const std::size_t checkersize = 20;

    // get glutwindow instance
    glutwindow& window = glutwindow::instance();

    // create a ppmwriter
    ppmwriter image (window.width(), window.height(), "./
checkerboard.ppm");

    // for all pixels of window
    for (std::size_t y = 0; y < window.height(); ++y)
    {
      for (std::size_t x = 0; x < window.width(); ++x)
      {
        // create pixel at x,y
        pixel p(x, y);

        example_math3d();
```

```
        // compute color for pixel
        if ( ((x/checkersize)%2) != ((y/checkersize)%2)) {
                p.rgb = color(0.0, 1.0, float(x)/window.height());
        } else {
          p.rgb = color(1.0, 0.0, float(y)/window.width());
        }

        // write pixel to output window
        window.write(p);

        // write pixel to image writer
        image.write(p);
      }
    }

    // save final image
    image.save();
  }

  // this method shows how to use the supplied classes for matrix, point
and vector
  void example_math3d()
  {
    // create some geometric objects
    math3d::point origin;
    math3d::point p0 ( 1.0, 5.0, -3.0 );
    math3d::vector v0 ( 2.0, 2.0, 1.0 );

    // some methods
    v0.normalize();

    // operator's
    math3d::vector v1 = origin - p0;  // difference between points is a
vector
    double z_component = v0[2];       // random access into components of
point/vector

    // you can create transformation matrices
    math3d::matrix rotate = math3d::make_rotation_x ( M_PI / 4.0 );    //
rotation matrix: 45 deg about x axis
    math3d::matrix scale  = math3d::make_scale      ( 1.0, 2.0, 1.0 ); //
scale y axis
    math3d::matrix invrot = math3d::inverse(rotate);

    // concatenate transformations and transform points or vectors
    math3d::vector tv0 = rotate * scale * v0;
    math3d::point  tp0 = rotate * scale * p0;
  }

private : // attributes
```

```
  // you may add your scene description here

};




int main(int argc, char* argv[])
{
  // set resolution and checkersize
  const std::size_t width = 400;
  const std::size_t height = 400;

  // create output window
  glutwindow::init(width, height, 100, 100, "CheckerBoard", argc, argv);

  // create a ray tracing application
  application app;

  // start computation in thread
  boost::thread thr(boost::bind(&application::start, &app));

  // start output on glutwindow
  glutwindow::instance().run();

  // wait on thread
  thr.join();

  return 0;
}
```