2D checkers

The get_color function for the PlaneChecker 2D texture is in Listing 30.4. Here, the outline width is specified as an absolute value in the *x* and *z* directions. The sets of lines parallel to the *x* and *z* axes have the same thickness.

In contrast, the outline widths for the other 2D checkers: ConeChecker, CylinderChecker, DiskChecker, RectangleChecker, and SphereChecker are specified as decimal fractions of the checker sizes in the respective directions. This seemed more sensible, because these checkers are all defined over finite intervals - either linear distance or angle. In addition, the vertical lines on the sphere and cone don't have a constant thickness. The user interface specifies the number of checkers in each direction. For example, with the CylinderChecker we specify the number of checkers around the cylinder in the angular range $[0, 2\pi]$, and the number in the vertical (y) direction in the interval $y \in [-1, +1]$. This is the y extent of a generic cylinder. The checker dimensions are computed from these values, and we can specify different line widths in each direction. For example, a line width of 0.1 is 1/10 of the width of the checker in the direction *perpendicular* to the line. For example, in the CylinderChecker, the width of the horizontal lines is specified as a fraction of the height of the checkers (in the y direction).

The 2D checkers can only be applied directly to generic objects, including a generic plane - the (x, z) plane.

The Chapter 30 download includes the SphereChecker class. You can use this as a model for implementing the other 2D checker textures.

I've included an image based on Figure 30.4, that shows all the checkers with outlines. You can use this, as well as Figure 30.4, for testing purposes.

Build functions

I've included build functions for all the images except Figure 30.3, which doesn't need a separate build function, and Figure 30.15, which I've left as an exercise. For Figure 30.15 you will have to implement a TorusChecker texture.

Figures 30.1, 30.6, 30.13, and 30.16 involve noise-based textures that are discussed in Chapter 31. The texture details are different because the original images were rendered a few years ago, and the code has been updated.

Ray traced images

I've re-rendered a number of these images in higher resolution. All color images are now in RGB.

Exercises

Exercise 30.10 is a load of rubbish. This is because the ShortShadeRec class inherits from ShadeRec. When it's constructor is called, C++ will also call the constructor of the base class ShadeRec, which contains all the data members. In fact, my code for the constructor explicitly calls the ShadeRec constructor:

```
ShortShadeRec::ShortShadeRec (World& w, const Point3D& local_hit)
  :       ShadeRec(w),
          local_hit_point(local_hit)
{}
```

This call has to be here because of the World argument. ShortShadeRec also has to inherit from ShadeRec so that it can be passed into the textures' get_color functions.

The result will therefore be an increase in rendering times!