Angel and Shreiner: Interactive Computer Graphics, Seventh Edition

Chapter 7 Solutions

7.1 The major problem is that the environment map is computed without the object in the scene. Thus, all global lighting calculations of which it should be a part are incorrect. These errors can be most noticeable if there are other reflective objects will now not show the reflection of the removed object. Other errors can be caused by the removed object no longer blocking light and by its shadows being missing. Other visual errors can be due to distortions in the mapping of the environment to a simple shape, such as a cube, and to errors in a two step mapping. In addition, a new environment map should be computed for each viewpoint.

7.2 This is an example of time domain aliasing. The wheels are spinning in time.As the wheels increase their speed, the frame rate is insufficient to avoid aliasing. The frequency of rotation then exceeds the Nyquist rate and is aliased as a negative frequency which is visible as the wheels rotating in the opposite direction. Because the frame rate is fixed, this error is difficult to avoid. We can use wheels without spokes but this solution may not be very good for stage coaches in westerns.

7.3 Consider a single sine wave that is sampled at a rate just over the Nyquist rate with the first sample where the sine is zero. The next sample will be slightly greater than zero, the following slightly greater (in magnitude) than the second, and so until we get samples about equal to the maximum of the sine. Subsequent samples will get smaller and smaller until we get to the minimum of the sine, and then the values will slowly increase. This pattern will repeat, each cycle taking multiple periods of the original sine. Visually, this pattern looks like a modulated sine wave. This visual appearance is due to us seeing the energy of the original samples and those of the first replica which appears just over the Nyquist rate. Our eye is a low pass but imperfect filter that lets both pass through. If we add the two sinusoids using the trigonometric identity we see the sum and difference frequencies which explains this modulation. The higher frequency (the sum) gives the inner frequency while the difference generates the slowly varying envelope. These patterns describe the Moire patterns we see when we move past two fences, one behind the other, when driving in a car.

7.4 Television is digital in the vertical direction and analog along each scan

line. Hence, aliasing errors only occur in the vertical direction and the pattern depends on the relationship between the number of scan lines (the vertical sampling interval) and the vertical spatial frequency content of the data. When someone wearing a striped tie or striped shirt changes, the angle between the repeated pattern and the scanlines change, thus changing the Moire patterns.
Another way to understand this effect, is to consider two rows of bars, one of which is horizontal and represents the scan lines of the television. Assuming the second is translucent, we can place it is over the first (the scan lines). If the second pattern consists of vertical bars, its spatial frequency relative to the scan lines is zero because we see constant values as we move vertically. As we rotate the second pattern, we see increasing spatial frequencies (proportional to the sine of the angle between the two patterns). When the second pattern is parallel to the first we have the highest relative frequency. Try this by making a transparency of vertical bars with a photocopier.

7.5 Once aliasing occurs, it is very difficult to remove because the data that has been aliased at another frequency is mixed with any unaliased data at that frequency. Thus, if we remove energy at that frequency, we remove both aliased and unaliased information. On the other hand, if we remove data at frequencies above the Nyquist frequency before sampling occurs, although we loose information, we are left with only unaliased data after sampling occurs.

7.6 If a surface has opacity α, a fraction $1 - \alpha$ of the light from behind it will pass through it and will be seen from the front. Now consider two surfaces with opacities α_1 and α_2. A fraction $1 - \alpha_1$ will pass through the first and this amount will be reduced by $1 - \alpha_2$ by passing through the second, leaving the same amount of light as would a surface with transparency $(1 - \alpha_1)(1 - \alpha_2)$ or equivalently the amount of light passing through a surface of opacity $1 - (1 - \alpha_1)(1 - \alpha_2)$.

7.7 The basic problem is that when we use subtractive colors and filters, the colors modulate each other and there is a multiplicative rather than additive relationship. Thus, if we have two CMY colors. (C_1, M_1, Y_1) and (C_2, M_2, Y_2), the resulting color is (C_1C_2, M_1M_2, Y_1Y_2). OpenGL supports this kind of blending by allowing us to set a source or destination blending factor that is itself a color, Thus if the source blending factor is the destination color and destination blending factor is zero, we can create a filter.

7.8 There are two issues. First, if we use α and $1 - \alpha$ we avoid problems of having colors and opacities exceeding 1 and being clipped. However, by using this choice, the order in which we composite multiple surfaces matters which would not make a difference if we used α and 1.

7.10 Suppose that we have three–dimensional texture memory. We can load this memory with our three dimensional voxel data. Next we can form a series of 100-200 parallel polygons facing the viewer that slice through the three–dimensional box of texture. We can map the texture onto these polygons as they are displayed. Using compositing, we can see all voxels. If we can do texture mapping in hardware, then have a real time display because we need only display a small number of polygons. The limitation of this technique is the large amount of texture memory that is required.

7.11 Suppose that the histogram of the image is a function $f(x)$ where x is the luminance. The lookup table formed from the function $\int_0^x g(x')dx'$ will create an image with a flat histogram. A simple discrete example is illustrative. Suppose we have a image which is 1024 x 1024 and has 256 values of luminance. If the image we want to create is to have a flat histogram, there should be 4096 pixels with each luminance value. If we have the histogram of the original image, we can use this histogram to find which original luminance values we must use to obtain the lowest 4096 values to assign to 0, the next lowest 4096 to assign to 1, and so on. The function that describes this process is the integral of the histogram curve.

7.12 Whenever we use regular patterns, we risk creating beat patterns or Moire effects. Random jitter avoids the problem. Note that in a mathematical sense we are no better off using jitter. However, our visual systems are very sensitive to regular patterns so that jittering makes the images appear to be better.

7.13 In the perspective projection, each row of squares will have a different size and thus each will appear as a pattern with a different frequency. Consequently, there will be different Moire patterns. In a parallel projection, once the checkerboard is oriented, the patterns are all the same.

7.14 There are a couple of ways to do this problem.Pre 3.1 OpenGL used the OpenGL function `glTexGen` to generate texture coordinates automatically. This function allowed the generation in either object space or eye space by computing texture coordinates based on distance from a plane specified by the application. This method can be done in a vertex

shader.Another option is to use a matrix to scale the texture coordinates appropriately for each parallelogram.

Consider the side of the cube determined by the plane $y = 1$. The projector through a point $(x, 1, z)$ on this plane is on a line from the origin that intersects the unit sphere $x_p^2 + y_p^2 + z_p^2 = 1$. Points on this line are of the form $(\alpha x, \alpha y, \alpha z)$. Using the fact that the line passes though the plane $y = 1$ yields $\alpha = \frac{1}{y_p}$ and thus $x_p = xy_p$ and $z_p = zy_p$. We can now use the equation of the sphere to solve for $y_p = \frac{1}{\sqrt{1+x^2+z^2}}$ and then x_p and z_p.

7.22 No. Consider a corner of the cube. If the top face is black, both the front and right must be white. But then when we cross from the front to the right, we will not have a change in color.

7 .23 Generally back-to-front rendering is nice because faces in front always paint over surfaces behind them. However, the final color is not determined until the front most object is processed. Suppose you have a lot of overlapping opaque objects. Then most of the rendering will have been wasted since only the final faces will determine the image. In applications such as ray tracers, a front-to-back rendering can be far more efficient as we can stop processing objects along a ray as soon as we encounter the first opaque object.

7.26 The basic idea is that the dot product of the normal and other vectors such as the light vector must remain unchanged by an affine transformation. In matrix terms, consider a vector a that is transformed to Ta. if we transform another vector b to Sb. We want

$$a \cdot \equiv Ta \cdot Sb$$

or

$$Ta^T Sb = a^T A^T Sb = a^T b = a^T Ib.$$

Hence

$$T^T S = I.$$

If a and b are the representations of vectors they need only be three dimensional and T can be the upper-left 3×3 submatrix of the model-view matrix.

7.27 Each time a new mipmap is created, we need 1/4 of the storage of the previous level. The upper bound is given by the series $1 + 1/4 + 1/16 + 1/64 + \ldots = 4/3$. Thus we need at most 1/3 more storage.