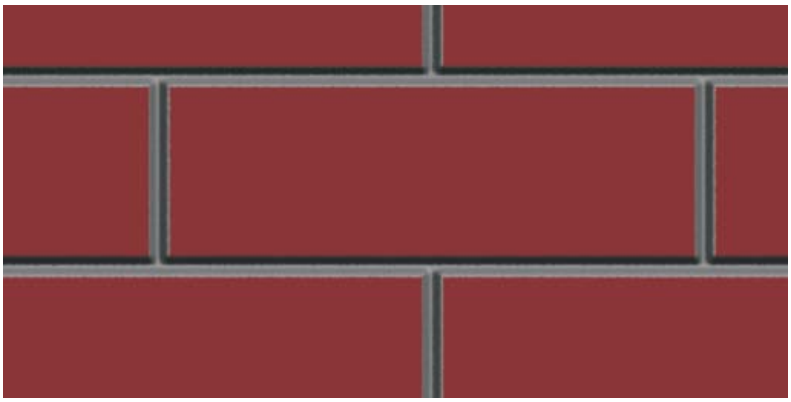

FIGURE 2.25 The bump-mapped brick texture.

to the `world` space. To avoid this problem, we could have transformed the surface point and normal vector into `shader` space, done the displacement there, and transformed the new normal back to `current` space, as follows:

```
point Nsh, Psh;
Psh = transform("shader", P);
Nsh = normalize(ntransform("shader", N));
Nsh = calculatenormal(Psh + Nsh * stbump);
Nf = ntransform("shader", "current", Nsh);
Nf = normalize(faceforward(Nf, I));
```

Note the use of `ntransform` rather than `transform` to transform normal vectors from one space to another. Normal vectors are transformed differently than points or direction vectors (see pages 216–217 of Foley et al. 1990). The second `ntransform` uses two space names to request a transformation from `shader` space to `current` space.

Example: Procedural Star Texture

Now let's try to generate a texture pattern that consists of a yellow five-pointed star on a background color `Cs`. The star pattern seems quite difficult until you think about it in polar coordinates. This is an example of how choosing the appropriate feature space makes it much easier to generate a tricky feature.

Figure 2.26 shows that each point of a five-pointed star is 72 degrees wide. Each half-point (36 degrees) is described by a single edge. The end points of the edge are a

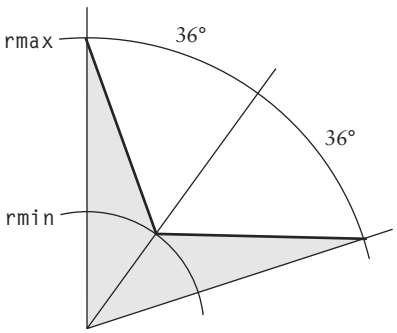

FIGURE 2.26 The geometry of a star.

point at radius `rmin` from the center of the star and another point at radius `rmax` from the center of the star.

```
surface
star(
      uniform float Ka = 1;
      uniform float Kd = 1;
      uniform color starcolor = color (1.0000,0.5161,0.0000);
      uniform float npoints = 5;
      uniform float sctr = 0.5;
      uniform float tctr = 0.5;
    )
{
      point Nf = normalize(faceforward(N, I));
      color Ct;
      float ss, tt, angle, r, a, in_out;
      uniform float rmin = 0.07, rmax = 0.2;
      uniform float starangle = 2*PI/npoints;
      uniform point p0 = rmax*(cos(0),sin(0), 0);
      uniform point pi = rmin*
          (cos(starangle/2),sin(starangle/2),0);
      uniform point d0 = pi - p0; point d1;
      ss = s - sctr; tt = t - tctr;
      angle = atan(ss, tt) + PI;
      r = sqrt(ss*ss + tt*tt);
```

At this point, the shader has computed polar coordinates relative to the center of the star. These coordinates `r` and `angle` act as the feature space for the star.

```
    a = mod(angle, starangle)/starangle;
    if (a >= 0.5)
          a = 1 - a;
```

Now the shader has computed the coordinates of the sample point `(r,a)` in a new feature space: the space of one point of the star. `a` is first set to range from 0 to 1 over each star point. To avoid checking both of the edges that define the "V" shape of the star point, sample points in the upper half of the star point are reflected through the center line of the star point. The new sample point `(r,a)` is inside the star if and only if the original sample point was inside the star, due to the symmetry of the star point around its center line.

```
    dl = r*(cos(a), sin(a),0) - p0;
    in_out = step(0, zcomp(d0^d1) );
    Ct = mix(Cs, starcolor, in_out);
    /* diffuse ("matte") shading model */
    Oi = Os;
    Ci = Os * Ct * (Ka * ambient() + Kd * diffuse(Nf));
}
```

To test whether `(r,a)` is inside the star, the shader finds the vectors `d0` from the tip of the star point to the `rmin` vertex and `d1` from the tip of the star point to the sample point. Now we use a handy trick from vector algebra. The cross product of two vectors is perpendicular to the plane containing the vectors, but there are two directions in which it could point. If the plane of the two vectors is the (x, y) plane, the cross product will point along the positive z-axis or along the negative z-axis. The direction in which it points is determined by whether the first vector is to the left or to the right of the second vector. So we can use the direction of the cross product to decide which side of the star edge `d0` the sample point is on.

Since the vectors `d0` and `d1` have z components of zero, the cross product will have x and y components of zero. Therefore, the shader can simply test the sign of `zcomp(d0^d1)`. We use `step(0,zcomp(d0^d1))` instead of `sign(zcomp(d0^d1))` because the `sign` function returns -1, 0, or 1. We want a binary (0 or 1) answer to the query "Is the sample point inside or outside the star?" This binary answer, `in_out`, is used to select the texture color `Ct` using the `mix` function, and the texture color is used to shade the sample point according to the diffuse shading model.

Figure 2.27 is an image rendered using the `star` shader.

Spectral Synthesis

Gardner (1984, 1985) demonstrated that procedural methods could generate remarkably complex and natural-looking textures simply by using a combination of sinusoidal component functions of differing frequencies, amplitudes, and phases. The theory of Fourier analysis tells us that functions can be represented as a sum