CS 465 Homework 8 Solutions

Part A

1. The fragment program has several input arguments. Which of these need to be interpolated by the rasterizer and which are constants?

Answer: $\mathbf{n}$, $\mathbf{v}_E$, and k_d are interpolated because we have a local viewer and spatially varying diffuse color. All other values (k_s, p, I, and $\mathbf{v}_L$) are constant.

Part B

1. Describe how to implement this using a texture map T of size 256×256

   Answer: There are many possible variations, which will affect the answers to some of the problems below. Here we give the most common answer: let $T(i,j) = round((2^{16}-1)(i/255)^j)$. In this case the dot product $\mathbf{v}_H \cdot \mathbf{n}$ varies from 0 to 1 along the rows of T and the exponent p varies from 0 to 255 along the columns.

2. Give pseudocode for the fragment program in a form similar to above.

   Answer:

   For our answer above, this becomes

   shade-fragment($\mathbf{n}$, k_d, k_s, p, I, $\mathbf{v}_L$, $\mathbf{v}_E$)
   $\mathbf{v}_H = \text{normalize}(\mathbf{v}_L + \mathbf{v}_E)$
   $r_d = k_d \max(0, \mathbf{v}_L \cdot \mathbf{n})$
   $i = 255 \max(0, v_H \cdot n)$
   $r_s = k_s$fetch-texture$(T, i, p)/(2^{16}-1)$
   return $I(r_d + r_s)$

3. Which parameters are varying and which are constant?

   Answer: $\mathbf{n}$, $\mathbf{v}_E$, k_d, k_s, p are varying. I and $\mathbf{v}_L$ are constant.

4. What is the value of $T(140, 110)$?

   Answer:

   Again, for our choice of T,

$$\begin{aligned} T(140,110) &= \text{round}((2^{16}-1)(140/255)^{110}) \\ &= \text{round}(1.483e^{-24}) \\ &= 0 \end{aligned}$$

5. What value will your fragment program return for $\mathbf{n} = \frac{1}{\sqrt{41}}[1;6;2]$, $\mathbf{v}_E = \frac{1}{5}[0;3;4]$, $\mathbf{v}_L = \frac{1}{\sqrt{17}}[1;0;4]$, $p = 3.6$, $k_s = 0.8$, $k_d = 0.2$, $I = 1$

   Answer:

   Since r_d does not depend on the texture map implementation, it will always be the same: $k_d\mathbf{v}_l \cdot \mathbf{n} = (0.2)(0.3409) = 0.06818$.

   For the specular component, we first compute the indices into our texture map:

$$\begin{aligned} \mathbf{v}_H &= \text{normalize}(\mathbf{v}_L + \mathbf{v}_E) \\ &= (0.1287; 0.3183; 0.9392) \\ i &= 255\mathbf{v}_H \cdot \mathbf{n} \\ &= 155.9984 \\ j &= 3.6 \end{aligned}$$

   fetch-texture will bilinearly interpolate from the nearest integer indices, namely $i_1 = 155$, $i_2 = 156$, and $j_1 = 3$, $j_2 = 4$ with weights $w_i = 0.9984$ and $w_j = 0.6$. This gives us:

$$\begin{aligned} \text{fetch-texture}(T,i,j) &= (1-w_i)(1-w_j)T(i_1,j_1) + w_i(1-w_j)T(i_2,j_1) \\ &\quad + w_j(1-w_i)T(i_1,j_2) + w_iw_jT(i_2,j_2) \\ &= (0.0016)(0.4)(14718) + (0.9984)(0.4)(15005) \\ &\quad + (0.0016)(0.6)(8946) + (0.9984)(0.6)(9179) \\ &= 11508.99 \end{aligned}$$

   which gives us

$$\begin{aligned} r_s &= k_s\text{fetch-texture}(T,i,j)/(2^{16}-1) \\ &= (0.8)(11508.9/65535) \\ &= 0.14049 \end{aligned}$$

so our computed shading value is $I(r_s + r_d) = 0.20867$.

6. What is the relative error of your answer from part (5) compared to the true value?

   Answer:

   Plugging in the values given into the original shade-fragment program (without the texture speedup) gives us a return value of 0.20457. Taking our answer from (5), we get the relative error to be $\frac{0.20867-0.20457}{0.20457} = 0.02004$, or about 2%.

Part C

Now we wish to implement a more advanced reflection model, the Ward model. In this model the reflected color is defined by the equation:

$$r_s = k_s \sqrt{\frac{\cos\theta_E}{\cos\theta_L}} \frac{e^{-\left(\frac{\tan\alpha}{m}\right)^2}}{m^2}$$

where θ_E is the angle between $\mathbf{v}_E$ and $\mathbf{n}$, θ_L is the angle between $\mathbf{v}_L$ and $\mathbf{n}$, α is the angle between $\mathbf{v}_H$ and $\mathbf{n}$, and $m \in [0, 1]$ is the parameter that controls the size of the highlight.

1. Describe how this model can be implemented by a similar approach using texture maps. Use the simplest texture maps possible (that is, use 1D maps in preference to 2D maps and fewer maps in preference to more maps) while avoiding the computation of exponentials, square roots, and inverse trigonometric functions in the shader.

   Answer:

   We first make the observation that $\mathbf{v}_E \cdot \mathbf{n} = |\mathbf{v}_E||\mathbf{n}| \cos\theta_E$ (and similarly for θ_L and α). Since our eye and normal vectors are already normalized, we can use the dot product to directly calculate the cosine of the angles. Moreover, since we will never see eye and light rays below the plane perpendicular to the normal vector, these cosines will always be in the range $[0, 1]$.

   To begin with, we notice that $\sqrt{\frac{\cos(\theta_E)}{cos(\theta_l)}}$ can be changed to $\frac{\sqrt{\mathbf{v}_E \cdot \mathbf{n}}}{\sqrt{\mathbf{v}_L \cdot \mathbf{n}}}$. Since our dot products share the same domain, we can use a single 256×1 1D texture map to store the square root computation like so:

   $$T_1(i, 0) = \text{round}((2^{16} - 1) * \text{sqrt}(i/255))$$

   which stores the square roots of sample points between 0 and 1. Note that if we want to do this with a 2D map we need to be careful about clamping: $\frac{\cos\theta_E}{\cos\theta_L}$ can go to infinity as θ_L approaches $\pi/2$, so we need to be extremely careful about clamping at large values so that each $T(i, 0)$ does not exceed $2^{16} - 1$.

As for the second term, we observe that:

$$\begin{aligned}(\tan(\alpha)/m)^2 &= \sin^2(\alpha)/(m^2\cos^2(\alpha)) \\ &= (1-\cos^2(\alpha))/(m^2\cos^2(\alpha)) \\ x &= (1-(\mathbf{v}_H\cdot\mathbf{n})^2)/(m(\mathbf{v}_H\cdot\mathbf{n}))^2\end{aligned}$$

The problem is that x can be between $[0,\infty)$, so there is no way to map this to the finite domain of a texture map. One way around this is to create a 2D texture map, where the indices are $\cos\alpha$ and m, like so:

$$T_2(i,j) = \text{round}((2^{16}-1)*\exp(-((1-(i/255)^2)/(j/255*i/255)^2))$$

Another way is to carefully clamp your inputs to never exceed a prescribed bound. Here we present a third way that does not require any clamping of values.

Take x from above. It can take on any value in $[0,\infty)$. We would like to map it to a finite domain which we can then use as a texture map. Take $y = 1/(1+x)$. Now, $y \in (0,1]$, so we can now create a 1D texture map

$$T_2(i,0) = \text{round}((2^{16}-1)*e^{-(255-i)/i}$$

noting that $x = (1-y)/y$. Thus, we can use only two 1D maps and eliminate all expensive mathematical operations without any arbitrary clamping required.

2. Give pseudocode for the fragment program in a form similar to above.

Answer:

We give the pseudocode for the more common case of a 1D map and a 2D map. Converting it to use the two 1D maps as described above is straightforward.

shade-fragment(**n**, k_d, k_s, p, I, $\mathbf{v}_L$, $\mathbf{v}_E$)
  $\mathbf{v}_H = \text{normalize}(\mathbf{v}_L + \mathbf{v}_E)$
  $r_d = k_d \max(0, \mathbf{v}_L\cdot\mathbf{n})$
  veInd $= 255\max(0,\mathbf{v}_E\cdot\mathbf{n})$
  vlInd $= 255\max(0,\mathbf{v}_L\cdot\mathbf{n})$
  cosTrm = fetch-texture$(T_1, \text{veInd}, 0)$/fetch-texture$(T_1, \text{vlInd}, 0)$
  alphaInd $= 255\max(0,\mathbf{v}_H\cdot\mathbf{n})$
  eTrm = fetch-texture$(T_2, \text{alphaInd}, 255*m)/(2^{16}-1)$
  $r_s = k_s *$ cosTrm $*$ eTrm$/(m*m)$
  return $I(r_d + r_s)$

3. Which parameters are varying and which are constant?

   Answer:

   It is intended that k_s and m can vary over the surface just as in part B, but since that was unspecified any answer for m and k_s that was consistent with answers from (1) and (2) was accepted. Ideally, $\mathbf{n}$, $\mathbf{v}_E$, k_d, k_s, p are varying. I and $\mathbf{v}_L$ are constant.