Introduction to Computer Graphics	**Prelim 2 Solutions**
CS465 Fall 2007	**Monday, November 12, 2007**

1. With an alpha channel, our typical color is stored as a vector $C_A = (r_A, g_A, b_A, \alpha_A)$. The equation for compositing colored surfaces A over (B over C) for the red channel (identical for other channels) is:

$$r = \alpha_A r_A + (1 - \alpha_A)(\alpha_B r_B + (1 - \alpha_B)\alpha_C r_C)$$

   We can see that in this equation, we never use the color values without first mulitplying by the respective alpha-channel value. As a result, we can pre-multiply alpha against each of the colors to give us color vectors of the form: $C'_A = (\alpha_A r_A, \alpha_A g_A, \alpha_A b_A, \alpha_A)$. With this our compositing color equation reduces to:

$$r = r'_A + (1 - \alpha_A)(r'_B + (1 - \alpha_B)r'_C)$$

   Pre-multiplying alpha assures that we only have to compute the composite colors once, and this computation does not need to be repeated each time C_A is used. Second, premultiplying alpha allows for associativity so that ((A over B) over C) = (A over (B over C)).

2. Some possible answers:
   Pros:

   - Unlike painter's algorithm, we can render our fragments in any order, and the front fragment will always show.
   - The algorithm is very easy and quick to implement. We only need to compute and compare z-values in view space, and make simple comparisons between them.
   - Again, unlike painter's algorithm, it does not degrade when there exist cyclic spatial relationships between fragments.

   Cons:

   - Space intensive. We must store extra data for every pixel, and to allow for decent precision, the size of this data cannot be ignorably small.
   - We need to implement clipping planes so as to give z-buffer meaningful maximum and minimum values.
   - Extra computation must be performed in the rasterization step to interpolate the z-values.

- We must check every fragment when rendering a pixel, since we do not keep track of the ordering of the fragments.

3. The following is a set of valid acceleration techniques. Many of these are listed in the Ray Tracing II class notes:

   - Bounding volumes. Create larger easy to intersect volumes that envelop some group of triangles. If a ray does not intersect the bounding volume, it cannot intersect any of the triangles.
   - Regular space subdivision. Split the entire space into evenly spaced subdivisions. And
   - Hierarchies of boudning volumes. Creates bounding volumes of bounding volumes adaptively to distribute triangles in bounding volumes evenly.
   - k-d Trees. Subdivides space adaptively like hierachies of bounding volumes.
   - Barycentric coordinates for triangle intersection can avoid exit early from intersection calculation the ray cannot intersect due to it's value in any one (or combination) of the coordinates.
   - Back-face culling. Ignore surfaces with normals in the same direction as traced ray.

4. A glossy reflection is one that has a shiny but glazed appearance; that is, there is a reflection, but the reflection is muddled. Glossiness often occurs when we have a reflective surface, but that surface is not perfectly smooth. Instead, the material at a microscopic level is bumpy/mountainous/ridged and point normals tend to vary slightly from what appears macroscopically. We can model this variation by "jittering" each point's normal through a random perturbation. This will also perturb the reflectance ray slightly, and neighboring points may have reflectance rays in dissimilar directions. By jittering the point normal several times and taking the average colors of these points, we can obtain a glossy reflection rather than a clear mirror-like one. The random perturbations can be included by perturbing the normals or the reflected rays themselves, so long as the changes only affect the direction and not the starting point of the reflecting rays.

5. (a) Recall that the formula that we need to use for Beer's law is:

$$I(s) = I_0 e^{-sa}$$

   where I(s) is the intensity of the outgoing light at distance s from the interface, I_0 is the intensity of the light at the interface, and a is some attenuation constant for the material.

For this problem we will show the equations for the attenuation constants of R of RGB but note that a_R,a_G,and a_B may be different for a single material. Let I_{r0} be the inital R value for the intensity of the initial white light. Then we know for attenutation contsant $a_{r,water}$ that after passing through the water the light's intensity is:

$$I_{water} = I_{r0}e^{-h_{water}a_{r,water}}$$

This water value is then used as the initial intensity for Beer's law again on the glass. Let the attenuation consant for the glass be $a_{r,glass}$, then:

$$I_{final} = I_{water}e^{-h_{glass}a_{r,glass}} = I_{r0}e^{-h_{water}a_{r,water}}e^{-h_{glass}a_{r,glass}}$$

If we'd like we can treat our initial light as a bright, maximum intensity white, in which case we can just omit the I_{r0} term since it would be equal to 1.

Note that the attenuation constant a is often set such that the equation for Beer's law is:

$$I(s) = I_0e^{-s\ln(a)}$$

Either equation is acceptable.

(b) No, the apparent color does not depend on the particular vertical order of the glass/water layers. As we can see in the equations above, each layer's affect simply becomes the multiplicative coefficient (I_r0) of the next application of Beer's law. Thus, the only thing that changes upon changing the order, is the order of the multiplicative terms, but since multiplication is commutative, this presents no problem.

This is also true when the incident view direction is non-normal. That is, the apparent color still does not depend on the vertical order of the glass/water layers, even when the view direction is non-normal. One might worry that the distance traveled within one of the layers depends on which order it is entered, but using Snell's law we can show that this is not the case. Let η, η_w, and η_g be the three refractive indices of air, water, and glass respectively. Also, let θ, θ_w, and θ_g be the respective angles with respect to the normals. From Snell we know upon entering the water:

$$\eta\sin(\theta) = \eta_w\sin(\theta_w)$$

Since the layers of water (and glass) are all parallel with one another, the angle the ray makes with the normal on one side of the layer is the same angle it makes when exiting the other side. Thus, between the water and glass, we can reapply Snell's law:

$$\eta_w\sin(\theta_w) = \eta_g\sin(\theta_g)$$

but then we also know:

$$\eta\sin(\theta) = \eta_g\sin(\theta_g)$$

This means that, perhaps surprisingly, the angle taken by the light ray in each layer is independent of the order of the layers, and only depends on the original incident angle and the ratio of refractive indices. We also know that the distance traveled in each layer can be computed by:

$$d = \frac{h}{\cos\theta}$$

Thus, the traveled in each layer is independent of the order of the layers, and combining this with the first part of the question, we know that the apparent color too will be independent.

Note: Due to some ambiguity in the phrasing of the question, one might interpret it as asking whether there is a difference in the apparent color of the perpendicular and non-perpendicular white light rays. There IS a difference in appearance because the light ray will travel in each layer longer, and so the attenuation effects will have a greater impact. Full credit was also given to students who correctly answered the question in this way.

6. The following are some advantages of Perlin noise over image-based texture mapping:

- As opposed to large 2-D texture maps that must be stored in memory for texture-mapping, Perlin noise simply requires a few random seeds.
- Perlin noise is essentially continuous and can be implemented on smaller and smaller scales such that zooming in will not cause pixilation as it can in texture mapping.
- Using the same Perlin noise function on different surfaces in the same image will not simply reuse the same image twice as if often the case with 2-D texture mapping.
- Perlin noise can with little added effort allow time as an extra input so that the textures can be animated. Texture maps would require each frame in the animation to be explicitly defined.
- Similar to animation, making slight changes and tweaking the appearance of Perlin noise simply requires tweaking a few of the parameters whereas slight changes in texture maps often require entire 2-D texture maps to be created.
- Perlin noise functions can be defined 3-D (x,y,z space) so that nothing need be done to map the Perlin Noise onto any geometry. 2-D texture maps must be explicitly fit onto geometries, which can be difficult for stranger shapes.