Computer Graphics Comprehensive Exam

Computer Science Department
Stanford University
Fall 2005

NAME:

Note: This is exam is *closed-book*.

The exam consists of 5 questions. Each question is worth 20 points. Please answer all the questions in the space provided, overflowing on to the back of the page if necessary.

You have 60 minutes to complete the exam.

1. [20 points] Computer graphics definitions.

Define in a few sentences each of the following computer graphics terms. Some of these terms may be used in other fields, so be sure to give the computer graphics meaning.

1A [5 points] Control points.

Continuous functions, curves and surfaces may be formed from a discrete set of points. These points are called the control points. Control points are used to for Bezier and B-Spline curves for example.

1B [5 points] Depth cueing.

Depth cueing is a technique for modulating the intensity or saturation of a color as a function of depth.

Depth cues are any technique used to provide information about depth. This is a more general usage.

1C [5 points] Tessellation.

Tesselation is a technique for converting a smooth surface into a set of polygons, quadrilaterals or triangles.

Tesselation also refers to tiling the plane with polygons. This is a more general usage.

1D [5 points] Dithering.

Dithering is a technique for approximating continuous greys by making a pattern of black and white dots. Dithering can be generalized to work with color, to to use dots of different values.

2. [20 points] Window to viewport transformation.

Computer graphics systems implement abstractions for windows and viewports on top of the physical framebuffer. A framebuffer is a 2D array of pixels. For example, an SXGA framebuffer has size 1024 by 768. The pixel in the upper left hand corner of the framebuffer is (0,0) and the pixel at the lower right hand corner is (1023,767). A viewport is a rectangular subset of the framebuffer. For example, the user may place the viewport so its upper left hand corner is at (128,128) and it has a size of (512, 512). A window is a logical coordinate system defined by the user. For example, the user will say the window has an x range of (-1.0..1.0) meaning that the left edge of the viewport has coordinate -1 .0 and the right edge is 1.0. Similarly for y.

2A [10 points]. Suppose you have a window with coordinates (wxmin, wxmax,wymin, wymax) and a viewport with coordinates (vxmin, vxmax, vymin, vxmax). Derive a transformation that converts a point (x,y) in window coordinates to a point (x',y') in viewport coordinates.

x' = (x – wxmin) * (vxmax – vxmin)/(wxmax – wxmin) + vxmin

y' = (y – wymin) * (vymax – vymin)/(wymax – wymin) + vymin

2B [10 points]. The framebuffer in turn is stored in addressable memory. Memory addresses are linear; that is, 1-dimensional. Suppose the SXGA framebuffer is stored beginning at memory address `base`. Assuming the above window and viewport transformation, what is the transformation from window coordinates (x,y) to memory addresses?

addr = base + (1024 * y + x) * bytesperpixel

3. [20 points] Window systems.

We all use window systems with overlapping windows everyday. By overlapping we mean that there are many windows visible and they are stacked on top of each other. Front windows can partially cover windows behind them.

Windows are normally associated with processes. Processes are allowed to draw into the windows that they own, or that they have permission to draw into. The graphic system must prevent processes from drawing into windows unless they have permission to do so.

Suppose you are implementing a drawing engine for common primitives like triangles, lines, points, characters etc. Describe a technique for ensuring that the process can only draw into the visible parts of the windows that it has permission to draw into. State your assumptions about how windows are represented in the system. Present the reasons you think your solution is efficient.

This is an open-ended questions. Common correct answers included:

1) Create an off-screen buffer for each window and composite them in back to front order.

2) Assign a window-id to each pixel and only write to a pixel if the window-id assigned to the drawing process is equal to the window-id of that pixel.

3) Maintain a clipping polygon that describes the visible part of each window and clip each primitive to that polygon.

In all approaches, you needed to describe how you prevented drawing in regions outside your window.

4. [20 points] Color and Transparency.

In computer graphics RGB triplets are used to both model light itself and the interaction of light with materials. For example, a light source emits light with color (Re,Ge,Be). Light from different sources can be added together. For example, if we have two light sources, the light from the two sources may be combined additively (Re+Re',Ge+Ge',Be+Be').

When light interacts with a material, some of the light may be absorbed. As an approximation, we can model the material as a filter with transmission coefficients (Rf,Gf,Bf). The transmission coefficients must be in the range 0 to 1. If light with energy (Re,Ge,Be) interacts with a filter with transmission coefficients (Rf,Gf,Bf) the amount of transmitted light is (Re*Rf,Ge*Gf,Be*Bf).

4A [7 points]. Suppose you want to draw a picture consisting of a stack of *n* transmissive surfaces in front of a single emissive surface. Write a formula for the visible light given the amount of light emitted and the filter coefficients of the *n* surfaces.

R = Re (Prod Rf_i) where Rf_ is the red transmission coefficient for the ith surface.

4B [7 points]. Suppose each surface in the stack both emits light and transmits light. Write a formula for the amount of visible light.

R_i = Re_i (Prod_j<i Rf_j)

Attenuate the light from surface i by all the filters in front of it.

R = sum_i R_i

4C [6 points]. Suppose these surfaces are contained "within a pixel." That is, we consider stack of surfaces where each surface is the portion of the surface intersected by the pixel. Now, since surfaces have finite size, only a fraction of the pixel will be covered by the surface. Call that fraction α. Write a formula for the amount of visible light coming from a stack of transmitting and reflecting surfaces where each surface only covers a fraction of the pixel.

Update emitted light for each surface

Re_i' = α_i Re_i
Rf_i' = α_i Rf_i + (1-α_i)

And then use the formula in 4B

5 [20 points] Point inside triangle.

A modern rasterization system accepts as input triangles and produces as output a set of pixel fragments. The output pixels are all those inside the triangle. In order to do this, it needs an algorithm to determine whether a point is inside a triangle.

5A [15 points]. Suppose the triangle is given by 3 vertices (x1,y1), (x2,y2), and (x3,y3). Describe an algorithm that tests whether a point (x,y) is inside a triangle. Justify your algorithm.

1. Form the line equation for each edge of the triangle

$A\,x + B\,y + C < 0$ if the point x, y contains the interior of the triangle

$A = (y1 - y2)$
$B = (x2 - x1)$
$C = (x2*y1-x1*y2)$

These equations must be formed consistently by taking pairs of points as you move counterclockwise or clockwise around the triangle.

2. A point is inside the triangle if it is inside all three lines. A point is inside the half-space formed by a line if the line equation evaluates to a negative quantity.

5B [5 points]. A tricky aspect of point-inside-triangle algorithms are the boundary cases. For example, a point may be exactly on an edge. This is bad if multiple adjacent triangles are drawn, since points on the shared edges are drawn twice. Briefly describe how you would modify your algorithm to prevent points on the boundaries from being drawn multiple times. Make sure that the point is at least drawn once!

A tie-breaker must be introduced for points on the line. There are various ways to do this.

One of the most common is to test whether the half-space lines to the left or the right of the line. If the half-space lies to the left, points on the line are considered inside; if the half-space lies to the right, they are outside. The side of the half-space can be found by looking at the normal to the line, or the direction that the line moves (for example, $x2 > x1$). The tie-breaking rule needs to work for horizontal lines as well. For example, if the line is horizontal, then the upper half-space is inside, and the lower half-space is outside.