Name:

Final examination

Computer Graphics 1 (15-462)

Grade value: 22%. Total marks: 160

Duration: 3 hours

Answer all questions for full credit

Question 1: Fill algorithms (20)

(a) What is the difference between flood filling and boundary filling? (2)

Flood fill changes all connected pixels of a color to a new color. Boundary fill changes all connected pixels *not* of a certain color to a new color.

(b) What is the simplest implementation of filling? (1)

A recursive calling method.

(c) Why is this approach unsatisfactory? (1)

Because it requires a very large depth of recursion and therefore many function calls before any function is returned from - this may crash a computer.

(d) What is the difference between a 4 connected and an 8 connected fill algorithm? Illustrate with a diagram. (2)

4 connected only goes to pixels immediately to the sides or above and below. 8 connected goes to all eight pixels that touch the central pixel - including the diagonals.

(e) What implication is there to using either 4 or 8 connected fill? Consider what happens when the two algorithms reach a sloping line one pixel wide. (3)

4 connected regions will stop at lines of width one pixel. 8 connected could pass through.

(f) Span based filling is an alternative to the recursive fill algorithm. What is a span in flood filling terms and in boundary filling terms? (2)

In flood fill a span is a row of pixels of the same color.
In boundary fill a span is a row of pixels between pixels of the boundary color.

(g) What data structure is used to keep track of the spans in this algorithm and how does this improve on the recursive algorithm? (2)

The spans that still need to be filled are kept in a stack - data is simply moved on and off the stack to continue the processing and so no recursion is required - provided the stack is large enough the computer will not crash.

(h) Give an algorithm for a simple span based fill algorithm. Include the initialization step and how spans are added to and removed from the stack. (5)

```
Initially set current span to span containing the seed.
LOOP:
    Fill the current span.
    Go up one line:
        Push all spans connected to current span onto the stack.
    Go down one line:
        Push all spans connected to current span onto the stack.
    Remove span from top of stack - make it the current span.
    Repeat LOOP until the stack is empty.
```

(i) How is the above algorithm made to give 4 connected or 8 connected fill? (2)

For 4 connected fill the searches for connected spans on the rows above and below the current one end exactly above the ends of the current span. For 8 connected the searches extend one pixel beyond the ends of the current span.

Question 2: Scan conversion (30)

(a) What is scan conversion? (1)

Scan conversion is the conversion of ideal geometric primitives into their best pixel approximations.

(b) Write out real code for the simple Differential Digital Analyzer (DDA) algorithm for scan conversion of lines. (5)

```
void line ( int x0, int y0, int x1, int y1 ) {
    float y = y0;
    float slope = (float)(y1-y0)/(float)(x1-x0);
    int x;

    for ( x = x0 ; x <= x1 ; x++ ) {
        draw_pixel ( x, Round(y) );
        y += slope;
    }
}
```

(c) What assumptions does this code make about the ordering of the points and the slope of the line? (2)

The x0 point must be less than x1 and the slope of the line must be greater than -1 and less than 1.

(d) What can be done in the cases where these conditions are not true? (2)

If the x values are incorrectly ordered then simply swap the the end points.
If the magnitude of the slope is greater than 1 then reflect the results (loop over the y values instead of the x values).

(e) With a sketch illustrate which pixels are drawn for an example line with slope greater than 0 and less than 1. Do not make precise calculations, simply illustrate a general appearance. (2)

(f) It is possible to do the scan conversion calculations using entirely integer operations. What is the name of this algorithm? (1)

Bresenham Line Algorithm

(g) The algorithm is based on the implicit function of a line. Write down the implicit function of a line and define and explain the components of the equation. (4)

$F(\vec{P}) = 2\vec{N} \cdot (\vec{P} - \vec{P}_0)$

The function F is 0 on the line. $\vec{N}$ is the normal to the line (at right angles to it). $\vec{P}$ is the general point to be tested and $\vec{P}_0$ is a point on the line.

(h) How is the implicit function useful in determining which pixel to draw next? (2)

The value of the implicit function half way between the next two possible positions is calculated and its sign is used to determine which of the two pixels is drawn.

(i) Calculating the value of the implicit function still requires the calculation of a dot product - how is this avoided? Show your working. (4)

The implicit function is calculated incrementally, consider:

$$\begin{aligned} F(\vec{P}+\Delta) &= 2\vec{N}\cdot(\vec{P}+\Delta-\vec{P_0}) \\ &= F(\vec{P})+2\vec{N}\Delta \end{aligned}$$

There are only two possible Δs - (1,0) and (1,1), so they can be pre-computed.

(j) The code to do this integer line drawing is:

```
void draw_line ( int x0, int y0, int x1, int y1 ) {

    int x, y = y0;

    int dx = 2*(x1-x0), dy = 2*(y1-y0);

    int dydx = dy - dx, d = dy - dx/2;


    for ( x = x0 ; x <= x1 ; x++ ) {

        draw_pixel ( x, y );

        if ( d < 0 ) { d += dy; }

        else { y += 1; d += dydx; }

    }

}
```

Annotate this code to explain what each variable represents and what each command does. (6)

(k) How many integer operations are required per pixel in this code? (1)

Either two or three integer adds.

Question 3: Visibility algorithms (20)

(a) In the display of 3D scenes what is the visibility problem? (1)

The visibility problem is the calculation of which surface of the scene is actually visible at a given pixel, ie the calculation of the closest surface seen by a given pixel.

(b) Painter's algorithm solves the visibility problem by sorting the scene objects by depth. Write some very simple pseudo-code which implements Painter's algorithm, giving the range of each of the loops used. (5)

```
Sort objects by their depth.
Loop through the sorted objects starting with the furthest.
    Loop through the y extent of this object.
        Loop through the x extent of this line of the object.
            Draw the pixel.
```

(c) The main loop of this function is extremely simple - explain why it solves the visibilty problem. (1)

The code draws all of every object, but as it draws the closer objects later, these overwrite the previous drawn objects and the correct surface is displayed at a given pixel.

(d) Where is the complexity in this algorithm which makes it difficult to implement? Mention the most significant problem. (2)

The sorting of the objects is the most complex part of the algorithm. In particular when one object intersects another, the objects must be divided up - this is awkward to do.

(e) The z-buffer algorithm also solves the visibility problem. Explain its operation with pseudo-code, including its initialization step and the ranges of the loops. (5)

```
Initialize:
Loop through all y.
    Loop though all x.
        zbuf(x,y) = infinity.

Main algorithm:
Loop through objects (in any order).
    Loop through y range of object.
        Loop through x range of this line of the object.
            If object's z at (x,y) < zbuf(x,y)
                zbuf(x,y) = object's z at (x,y)
                image(x,y) = object's color at (x,y)
```

(f) Where might this algorithm have problems? (1)

Choice of the near and far clipping planes - they should be chosen to give the best possible z-buffer resolution.
OR If the depth complexity is high and there is lots of redundant shading.

(g) The z-buffer algorithm can also be used to compute shadows in 3D scenes without having to compute ray tracing. Explain how the algorithm can be altered to achieve this. (5)

Run the z-buffer algorithm as if each light source where a camera.
Store the generated z-buffer as the shadow buffer for that light.
Run the real z-buffer algorithm and when calculating a pixel's intensity:

```
    transform it into each camera's coordinate system
        test its z value against the value in the shadow buffer
```

Include a lighting component from this light source only if the z value equals the z buffer value.

Question 4: Viewing (20)

(a) What transformations are required to place a camera at any suitable position to view a scene? (2)

A translation and an angle axis rotation.

(b) What three (mainly common sense) considerations go into the positioning of the camera in a scene? (3)

It must be positioned so that it has a clear view of the important action, it must be pointing at the important action and it should be oriented so that vertical in the image is vertical in the scene.

(c) Describe a simple view specification scheme in which the three requirements that you listed above are met. (3)

Define a *lookfrom* position where the camera is to be placed, also define a *lookat* position, where the image should be centred, and also define *vup* the world vector that should be pointing straight up in the image.

(d) Describe how each of the above sections of the viewing scheme is implemented. Explain how each component of the transformations required can be calculated. Recall that the rotation matrix for an angle-axis rotation of θ about axis $\vec{v}$ is given by: $R = \vec{v}\vec{v}^\top + \cos\theta(I - \vec{v}\vec{v}^\top) + \sin\theta\vec{v}^*$ (6)

Translate by -*lookfrom* to put the focal point of the camera at the origin.
Rotate the $\vec{v}$ = *lookat* - *lookfrom* axis to the z-axis, remember to normalise $\vec{v}$:
the rotation axis, $\vec{a}$, is:

$$\vec{a} = \frac{\vec{v} \times \vec{z}}{\|\vec{v} \times \vec{z}\|}$$

the rotation angle is: $\cos\theta = \vec{v}\cdot\vec{z}$ and $\sin\theta = \|\vec{v}\times\vec{z}\|$ the rotation matrix is:
$R = \vec{a}\vec{a}^\top + (\vec{v}\cdot\vec{z})(I - \vec{a}\vec{a}^\top) + \|\vec{v}\times\vec{z}\|\vec{a}^*$
Rotate about the z-axis to get *vup* into the yz-plane, ie vertical in the image.
angle is $\arctan\frac{x}{y}$, for x, y, z as *vup* rotated by R above.

(e) When might this viewing scheme break down? Why? (2)

If, after its rotation by R, *vup* lies along the z-axis this viewing scheme will break down. This means that the camera is looking straight up (or down) and the rotation angle required is undefined - it is impossible to make *vup* vertical in the image.

(f) Explain what each of the three components of the equation that gives the angle-axis rotation matrix (from part (d)) achieve. (4)

The first part, $\vec{v}\vec{v}^\top$ is the projection onto the rotaton axis - this gives the part of the rotated vector that is unchanged by the rotation.
The second two parts should be considered together: the $(I - \vec{v}\vec{v}^\top)$ component is the projection of the vector to be rotated on to a plane to which $\vec{v}$ is normal and $\vec{v}^*$ is the projection onto that plane, rotated by $90°$. Multiplying by $\cos\theta$ and $\sin\theta$ gives the result of rotating by θ in that plane.
Adding these two parts together gives the rotated vector - one component that is unchanged plus the part that is actually affected by the rotation.

Question 5: Illumination (40)

(a) What is light and what gives it different colors? (2)

Light is electro-magnetic radiation transmitted by photons, color is simply a combination of wavelengths of EM radiation.

(b) Why do we encode light as only three numbers? (1)

The human eye has only three color receptors - RGB approximately correspond to the colors perceived by each type of receptor and thus RGB can represent (almost) any perceivable color.

(c) What would be two implications for television if it had to reproduce the whole spectrum of light? (2)

The reproduction of realistic looking images would be much more complex than the three electron guns or LED types used now. The transmission of the color information would also require much wider bandwidths.

(d) What is the difference between additive and subtractive color? (2)

Additive color is what we deal with on normal displays - the screen starts off as black and we add certain amounts of red, green and blue to give the desired result. Subtractive color is more applicable to painting and the use of pigments - we start off with white and take colors away from the white to produce the desired final result.

(e) What three components of illumination (not color) are used to calculate shading for an opaque surface? (3)

Ambient light, diffuse and specular reflectance.

(f) What two simple attenuation effects can be added to these illumination calculations? Explain the physical reasons for using these effects and give a simple equation if possible. (5)

Light and atmospheric. The brightness of a light on a surface at distance d from the light is reduced by $\frac{1}{d^2}$ - the area on which the power of the light falls increases by this much with distance - therefore the light's apparent brightness falls by this amount. The atmospheric attenuation blends the actual color to a "far" color as its distance from the viewer increases.

(g) If a surface is not totally opaque then transmission of light also occurs - what happens to the light rays if the refractive indices of the two media are different? (1)

The light ray is refracted, or bent by an angle (according to Snell's Law).

(h) It is very difficult to compute the above illumination terms using physical models and so approximations are used. A commonly used illumination equation is from Phong:

$$I = I_a k_a + f_{att} I_p (k_d \cos\theta + k_s \cos^n \alpha)$$

Explain all the values and terms in the above equation. (10)

I is the intensity seen by the camera.
The $I_a k_a$ term is for ambient lighting - the intensity and reflectance respectively.
The $f_{att} I_p$ is the light intensity reduced by an attenuation term due to distance of the surface from the light source.
The $k_d \cos\theta$ is the term describing the diffuse reflection, k_d is the surface's diffuse reflectance and the $\cos\theta$ angle accounts for the angle at which the surface is illuminated.
The $k_s \cos^n \alpha$ term accounts for specular reflection, k_s is the surface's specular reflectance and α is the angle between the the viewing angle and the angle at which true specular reflection would occur, the n value defines the size of the specular reflectance "beam", the larger it is the smaller the beam.

(i) The above equation refers to a single light source - how is the equation extended for multiple light sources? (2)

The equation is repeated for every light source - but only one component of the ambient light is added.

(j) How is color implemented using the above equation? (1)

The above equation refers to a single wavelength or color - it is repeated for each color required, usually red, green and blue.

(k) The above equation still does not produce very realistic images - what do the images look like and why are they unrealistic? (1)

The surfaces are uniformly colored or shaded - real surfaces have textures.

(l) Texture mapping can be used to improve images. There are several different ways of using it, name four of them. (4)

Texture mapping can vary the surface reflectance, the normal vectors - bump mapping, the reflectance coefficients - specularity mapping, light radience - environment mapping, geometry - displacement mapping, transparency - transparency mapping.

(m) Briefly explain how texture mapping is implemented. (4)

The texture is scanned and stored as an image. When generating the model each vertex of a textured surface has a texture coordinate as well as a 3D world coordinate. During rendering the texture (and geometric) coordinates for a pixel are calculated and the relevant surface parameters are calculated from the texture map values around those coordinates instead of using fixed values.

(n) One method of texture mapping generates the appearance of a rough surface on an actually planar surface. State how this is done and describe the anomaly that this approach produces. (2)

Instead of varying the surface intensity the texture varies the normal vectors by representing small upwards or downwards displacements from the true surface. The anomaly is that although the surface appears rough - in silhouette it will be look smooth, because it is only the shading calculations that are altered.

Question 6: Ray tracing (30)

(a) Explain how ray casting works. (3)

For each pixel in the scene a ray is shot into the scene. This ray is intersected with all the surfaces in the scene and the first surface that it hits is found. This surface is shaded using an equation similar to that given in the previous question to calculate the color that the pixel should be.

(b) Explain how recursive ray tracing extends ray casting. (2)

When the first intersection of a ray has been found, instead of simply using an equation to get the color, further rays are fired off from that point to calculate the correct shading.

(c) Typically four different kinds of rays are used during recursive ray tracing. What are these four types of ray and what are their functions? (8)

An eye ray brings light directly to the image (or eye) - it is the origin of all other rays.
A shadow ray brings light directly from a light to a surface - it finds if that light contributes any light to the illumination of that point on the surface.
A reflection ray is a ray reflected from a surface used to calculate the specular reflection component of the illumination of that surface.
A transmission ray is a ray passing through a transparent object, it is subject to refraction and calculates the illumination component of that surface point due to transmitted light.

(d) Three of the above rays cause more rays to be generated when they hit a surface, which of the rays doesn't? Explain why. (2)

Shadow rays do not generate further rays when they intersect a surface - if a shadow ray intersects a surface before it reaches the light for which it was aimed then the ray ends and no component for that light is added into the illumination equation for that point.

(e) One of the simplest ray intersection calculations is that for a sphere - solve this intersection problem. Show all your working and give the solution in its simplest possible form. Recall that one of the solutions of a quadratic equation $ax^2 + bx + c = 0$ is $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$. (5)

The implicit function for a sphere is: $x^2 + y^2 + z^2 - r^2 = 0$.
The parametric equation for a ray is: $x = x_0 + tv_x$, $y = y_0 + tv_y$, **and** $z = z_0 + tv_z$.
At the intersection the x, y, z **values must be equal, therefore put the ray equations into the implicit function for the sphere and solve for** t:

$$(x_0 + tv_x)^2 + (y_0 + tv_y)^2 + (z_0 + tv_z)^2 - r^2 \;=\; 0$$

Multiplying out gives:

$$t^2(v_x^2 + v_y^2 + v_z^2) + 2t(x_0v_x + y_0v_y + z_0v_z) + x_0^2 + y_0^2 + z_0^2 - r^2 \;=\; 0$$

Assuming that the direction of the ray is normalized, we have $a = 1$, $b = 2(x_0v_x + y_0v_y + z_0v_z)$ **and** $c = x_0^2 + y_0^2 + z_0^2 - r^2$. **Therefore:**

$$t \;=\; \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

(f) There are five possible outcomes for the values of t (in terms of the number of solutions, and the signs of the solutions), list them and explain the physical significance of each. (10)

t **could could have no real solution - the ray completely misses the sphere.**
t **could have one solution (the** $\sqrt{}$ **is zero) - the ray just grazes the sphere.**
Both values of t **could be negative - the intersections are both behind the camera and are therefore of no interest.**
One t **value is positive and one is negative - the focal point of the camera is inside the sphere - one intersection is in front of the camera and one is behind, take the positive value of** t.
Both t **values are positive - both intersections are in front of the camera so take the smaller value of** t.