1. [Total: 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]. Image composition.

Image composition is a method for combining two or more images into a single image. The combination may be done with a variety of operators including blend, over, sum, logical or, etc.

1B [5 points]. Diffuse reflection.

In diffuse reflection, incident light is scattered equally in all directions. This holds for all incoming directions.

1C [5 points]. Key-frame animation.

To animate an articulated character, the character is put into a particular pose at a sequence of times. The pose of the character at inbetween times are created by interpolating the parameters that control the pose. Key-frame animation can also be done in other ways. For example, different shapes may be assigned to different times and interpolated in a variety of different ways.

1D [5 points]. Hierarchical modeling.

A hierarchical model is a model that is composed of other models. In computer graphics this is represented as a DAG. The nodes consist of a transformation and pointers to other models. Hierarchical models are used to model parts made of subparts. For example, a body is comprised of a torso, two legs and two arms.

2. [20 points] Transformations.

Computer graphics relies heavily on 3D transformations. Most common are the geometric transformations such as rotations, translations, scales, etc. In the following, T(dx,dy,dz) refers to a translation by (dx,dy,dz), Rx(a) refers to a rotation about the x-axis by a degrees, Ry(a) and Rz(a) refer to rotations about the y- and z-axis, respectively.

The order of transformations may matter. Also, sometimes the order may be rearranged, but the arguments will change. Describe whether the following statements are true or false.

T(1,0,0) Rx(45) = Rx(45) T(1,0,0)? **FALSE**

T(1,0,0) T(0,2,0) = T(0,2,0) T(1,0,0)? **TRUE**

Rx(45) Ry(30) = Ry(30) Rx(45)? **FALSE**

Rx(45) Rx(30) = Rx(30) Rx(45)? **TRUE**

T(1,0,0) Rz(180) = T(-1,0,0) Rz(0) **FALSE**

Transformations have inverses. State whether the following formulas for the inverse transformations are true or false.

$Rz(45)^{-1}$ = Rz(-45)? **TRUE**

$Rz(180)^{-1}$ = Rz(180)? **TRUE**

$T(1,1,1)^{-1}$ = T(1,1,1)? **FALSE**

$[Rz(45)\ T(1,0,0)]^{-1}$ = T(1,0,0) Rz(45)? **FALSE**

$[Rx(45)\ Ry(30)]^{-1}$ = Ry(-30) Rx(-45)? **TRUE**

3. [20 points] Ray tracing.

One of the most general methods for rendering is ray tracing. At the core of a ray tracer is a procedure to find ray-surface intersections. The inputs to the procedure are a ray and a formula for a surface, the output is the point of intersection.

Assume is ray is given by the following parametric equations:

$$x = x0 + t * x1$$
$$y = y0 + t * y1$$
$$z = z0 + t * z1;$$

(x0,y0,z0) is the origin of the ray, and (x1,y1,z1) is the direction of the ray. As t increases, the points on the ray move from the origin along the direction.

Work out a procedure for computing the point of intersection of a ray with a cone. A cone is given by the following implicit function $x^2+y^2-z^2 = 0$. In general, a ray intersects a cone in two points. Return the closest point in the direction of the ray.

First, substitute the ray equation into the implicit equation for the cone. This yields the following formula:

$$(x0+t*x1)^2+(y0+t*y1)^2-(z0+t*z1)^2$$

$$= (x0^2+y0^2-z0^2)+2*(x0*x1+y0*y1-z0*z1)*t+(x1^2+y1^2-z1^2)*t^2$$

$$= a * t^2 + b * t + c = 0$$

This equation may be solved using the quadratic formula.

$$t = [-b +/- sqrt(b^2-4*a*c)] / (2*a);$$

The discriminant $D = sqrt(b^2-4*a*c)$ determines the nature of the solution. If $D < 0$, there are no real roots and the ray does not intersect the cone. If $D == 0$, there is a single root and the ray is tangent to the cone. If $D > 0$, there are two intersections.

The smallest, positive real root represents the closest intersection.

To find the point of intersection, t is plugged into the equation of the ray,

5. [20 points] Rasterization.

In your new job as chief graphics software architect, you receive a request to write a procedure to draw the following curve:

$$y^2 = x^3$$

To simplify the problem, consider only positive y values. This procedure should draw **n** pixels by stepping along the curve starting at the origin. Each pixel should be under the curve, or on the curve. No pixels should be above the curve. For example, (0,0) is on the curve, whereas (0,1) is above the curve; so we would choose (0,0) as the nearest pixel to the curve. The next few points would be (1,1), (2,2), (3,3), (3,4), (3,5), … The procedure should ensure that each point is connected (that is, is adjacent) to the previous point. The procedure should also be written as efficiently as possible. That is, use no multiplications except by constants in the inner loop. To draw the curve, write down an implicit function that defines the set of points on the curve, and then incrementally trace out the curve. Work out the math, and then fill in the following template:

```
drawcurve(int n)
{
  int x = 0;
  int y = 0;
  int x3 = x2 = 0;   // keep track of x*x and x*x*x
  int y2 = 0;        // keep track of y*y

  for(i=0; i<n; i++) {
    drawpoint(x,y);

    y2 += 2*y + 1;
    y  += 1;         // always move up

    if( y2 < x3 ) { // if above curve, move right

      x3 += 3*x2 + 3*x + 1;
      x2 += 2*x + 1;
      x  += 1;

    }
  }
}
```

4. [20 points] Polygonal Meshes.

Polygonal meshes are widely used in computer graphics. A polygonal mesh consists of vertices that define the positions of 3D points, and topological information that organize the points into polygons. There are many different types of mesh data structures, depending on the operations that need to be performed.

1. Describe a simple data structure to store a triangular mesh so that vertices are not duplicated. In your description, define C structs for vertices and triangles.

A simple data structure is

```
struct Vertex { float x, y, z; };

struct Triangle { struct Vertex *v1, *v2, *v3; };
```

2. Refine the data structure described above so that neighboring triangles may be found in constant time.

This can be done by modifying the above data structure to include pointers to the triangles across each edge.

```
struct Triangle {
   struct Vertex   *v1, *v2, *v3;
   struct Triangle *t1, *t2, *t3;
};
```

It important to define what triangle is adjacent across what edge. For example, `t1` is the triangle across the edge defined by `v2` and `v3`.