Theory Quiz Questions

Quiz Question 1: Ray construction formula

Consider the ray construction formula discussed in the lecture:

$$\mathbf{d} = -N\mathbf{n} + W\left(\frac{2c}{nCols - 1} - 1\right)\mathbf{u} + H\left(\frac{2r}{nRows - 1} - 1\right)\mathbf{v}$$

What is the effect of increasing the value of N?

Select one:

- a. Decrease the brightness of the scene
- b. None of the others
- c. Zoom out of the screen (i.e. the objects in the scene appear smaller)
- d. Distort the scene vertically
- e. Zoom into the screen (i.e. the objects in the scene appear larger)

Quiz Question 2: Ray-Plane intersection

What is the first intersection point p(t_0) of the ray $p(t) = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix} + t \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$

with the plane given by the equation: $2x + y - 4z = 4$

Select one:

- a. $\begin{pmatrix} 2 \\ 8 \\ 2 \end{pmatrix}$
- b. $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
- c. There are no points of intersection.
- d. $\begin{pmatrix} 1 \\ 6 \\ 1 \end{pmatrix}$
- e. None of the others.

Quiz Question 3: Ray-Disk intersection

Given is a disk in the xy-plane with radius r and centered at the origin. A ray with the equation $\mathbf{p}(t) = \mathbf{e} + \mathbf{tv}$, where $\mathbf{e} = (e_x, e_y, e_z)^T$ and $\mathbf{v} = (v_x, v_y, v_z)^T$, is intersecting the disk at point $\mathbf{p}(t_0)$. Which of the following statements is *correct*? You may assume that $e_z \neq 0$, i.e. the start point of the ray does not lie in the xy-plane.

Select one:

a. $t_0 = \frac{e_z}{v_z}$ and $(e_x + t_0 v_x)^2 + (e_y + t_0 v_y)^2 + (e_z + t_0 v_z)^2 \leq r^2$

b. $t_0 = 0$ and $(e_x + t_0 v_x)^2 + (e_y + t_0 v_y)^2 + (e_z + t_0 v_z)^2 \leq r^2$

c. $t_0 = -\frac{e_z}{v_z}$ and $(e_x + t_0 v_x)^2 + (e_y + t_0 v_y)^2 \leq r^2$

d. None of the others

e. $t_0 = -\frac{e_z}{v_z}$ and $(e_z + t_0 v_z)^2 \leq r^2$

Quiz Question 4: Ray-Sphere intersection

Consider two spheres: Sphere A has a radius of 3 and is centered at (3, 2, 1). Sphere B has a radius of 2 and is centered at (2, 5, 1). For a ray starting at (-2, -3, 0), and travelling in the direction (2, 1, 0), which of the following statements is true:

Select one:

a. Only sphere A is intersected by the ray.

b. Neither sphere is intersected by the ray.

c. Both spheres are intersected by the ray, sphere A is intersected first.

d. Both spheres are intersected by the ray, sphere B is intersected first.

e. Only sphere B is intersected by the ray.

Quiz Question 5: Ray intersects with transformed sphere

Given is a sphere with radius 1 centered at the origin. The sphere is transformed by the matrix $M = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. What is the first point of intersection of the ray $p(t) = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} + t \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$ with the transformed sphere?

Select one:

- a. The intersection point is $p(t_0)$ where $t_0 = 1$
- b. None of the others
- c. The intersection point is $p(t_0)$ where $t_0 = 2 - \frac{1}{4}\sqrt{5}$
- d. The intersection point is $p(t_0)$ where $t_0 = 2$
- e. The intersection point is $p(t_0)$ where $t_0 = 2 - \frac{1}{2}\sqrt{3}$

Programming Quiz Questions

Quiz Question 6: Ray construction

In this exercise you need to complete the code fragment below defining primary rays:

```
for(int r=0; r<windowHeight; r++) {
  for(int c=0; c<windowWidth; c++) {
    // construct ray through (c, r) using u,v,n and H,W
    //  Please ONLY MODIFY the values for dx, dy, dz (they are of type 'double')
    //  NOTE: C/C++ performs integer division if both arguments of a division are integers
    //        If you want to have a floating point division, one argument must be a floating
    //        point number, e.g., by using type conversion.

    dx = -n.x*N;
    dy = -n.y*N;
    dz = -n.z*N;
    Vector d = Vector(dx, dy, dz);
    // intersect ray with scene objects
    Hit hit = intersect(eye, d);
    // shade pixel accordingly
    Color color = shade(hit);
    glColor3f(color.r, color.g, color.b);
    glVertex2f((GLfloat)c, (GLfloat)r);
  }
}
```

Please compute the ray direction (dx, dy, dz) by modifying the equations for these values appropriately

Please use the following parameters:

W and H - window width and height (in units)

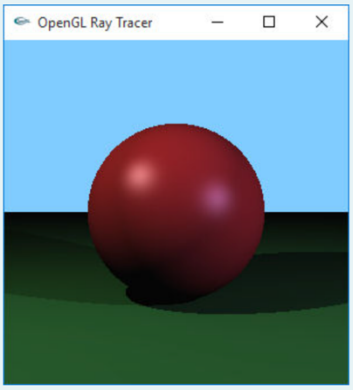
N - view plane distance from eye

windowWidth and windowHeight - size of display window in pixels (in lecture notes called nCols and nRows)

c and r - index of current pixel

u.x,u.y,u.z, v.x,v.y,v.z, n.x, n.y, n.z - uvn view coordinate system

Please paste the completed code fragment into the answer space. If your solution is correct you should get the following image:



Quiz Question 7: Ray-Plane intersection

In this exercise you need to complete the function below defining **Ray-Plane Intersection**:

```
double Plane::Intersect(Vector source, Vector d)
{
    /*=======================================================*/
    /* == Enter ray/plane intersection calculations here ==*/
    /* == Delete the line "t = -1.0;" and insert your solution instead ==*/
    /* == NOTE 1: If there is no ray-plane intersection set t = -1.0; ==*/
    /*=======================================================*/
        float t = -1.0;
        return t;
}
```

Note: The following variables are already defined for you to use:

float a - distance from origin to the plane

Vector n - plane normal

Vector: v1+v2 - adds the vectors v1 and v2 and returns resulting vector

Vector: v1-v2 - subtracts v2 from v1 and returns resulting vector

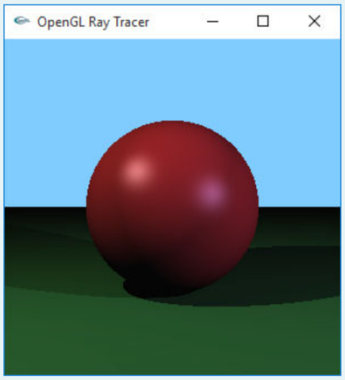
Vector: v1 * t - scales the vector v1 by t and returns resulting vector

Vector v.Scale(float a, float b, float c) - scales each component of the vector v

float v1.Dot(Vector v2) - returns the dot product of vector v1 and v2

Vector v1.Cross(Vector v2) - returns the cross product of vector v1 and v2

Please paste the completed code fragment into the answer space. If your solution is correct you should get the following image:



Quiz Question 8: Ray-Triangle intersection

In this exercise you need to complete the function below defining an intersection between a **Ray** and a **Plane with a triangular hole.** The plane normal **n** is provided and the hole in the plane is defined by three vertices A, B and C.

In the answer space is given a function for computing the intersection between a ray and a triangle. Modify this function such that it computes instead the intersection between a ray and a plane with a triangular hole. Your answer should have the following format:

```
double PlaneWithTriangleHole::Intersect(Vector source, Vector d)
{
        // modify calculation from ray-triangle intersection in the answer box below
        // and return correct value for t. If there is no intersection t should be -1.0
}
```

Note: The following variables and functions are already defined for you to use:

Vector A, B, C - coordinates of 3 vertices of the triangle

Vector n - plane normal

Vector: v1+v2 - adds the vectors v1 and v2 and returns resulting vector

Vector: v1-v2 - subtracts v2 from v1 and returns resulting vector

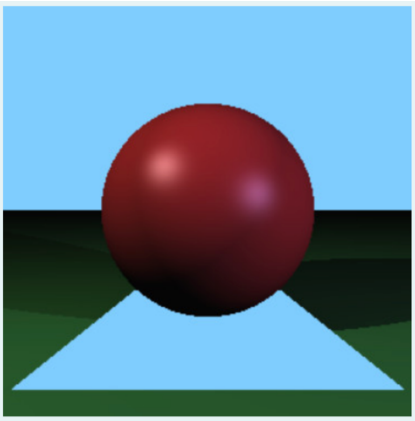
Vector: v1 * t - scales the vector v1 by t and returns resulting vector

Vector v.Scale(float a, float b, float c) - scales each component of the vector v

float v1.Dot(Vector v2) - returns the dot product of vector v1 and v2

Vector v1.Cross(Vector v2) - returns the cross product of vector v1 and v2

Please paste the completed code fragment into the answer space. If your solution is correct you should get the following image (note: the plane is the green surface at the bottom and the triangular hole appears blue (since we can see the background colour through it):



Provided structure code and hints:

```
// The code below computes the intersection with a triangle
// you need to change it so that you compute the intersection
// with a plane with  triangular hole
double PlaneWithTriangleHole::Intersect(Vector source, Vector d)
{
    float t = -1.0;
    float dn = d.Dot(n);
    if (dn == 0)    // plane and ray parallel
        return -1.0;
    t = (A - source).Dot(n) / dn;
    Vector p = source + d.Scale(t,t,t);
    float t1, t2, t3;
    t1 = (p - A).Cross(B - A).Dot(n);
    t2 = (p - B).Cross(C - B).Dot(n);
    t3 = (p - C).Cross(A - C).Dot(n);

    if (t1 < 0 && t2 < 0 && t3 < 0) return t;
    if (t1 > 0 && t2 > 0 && t3 > 0) return t;
    return -1.0;
}
```

Quiz Question 9: Ray-transformed sphere intersection

In this exercise you need to complete the functions below defining a **Ray-Sphere Intersection** for a transformed sphere. The sphere is first scaled and then translated. **NOTE: please complete both the intersection calculation and the normal calculation:**

```
double Sphere::Intersect(Vector source, Vector d)
{
        // Compute the intersection of a ray with a TRANSFORMED sphere
        // You can assume that for a transformed object the vectors 'scaling' and 'translation" are defined.
        // where (scaling.x,scaling.y,scaling.z) are the scale factors in x,y, and z direction
        // and (translation.x,translation.y,translation.z) is the translation vector

        // STEP 1: Transform the ray, compute the transformed ray start point and direction
        source=??
        d=??


        // STEP 2: Compute intersection of the transformed ray with the sphere
        // A, B, and C are the parameters of the quadratic equation for finding the
        // ray intersection parameter t (see "Ray Tracing" lecture notes)
        float A = ??
        float B = ??
        float C = ??

        float t; // the parameter t for the closest intersection point or ray with the sphere. If no intersection t=

        // BEGIN SOLUTION RAY-SPHERE INTERSECTION
        // ================================================================
        // == Delete the line "t=-1.0;" and insert your solution instead ==
        // == NOTE 1: If there is no ray-shere intersection set t=-1.0   ==
        // == NOTE 2: Use C notation so that the code runs in Coderunner ==
        // == NOTE 3: You might want to use the sqrt() function          ==
        // ================================================================
        t=-1.0;
        // END SOLUTION RAY-SPHERE INTERSECTION

        return t;
}
```

```
Vector Sphere::Normal(Vector p)
{
        // STEP 3: Calculate the correct normal for the transformed sphere
        // replace the existing code below and use the values 'scaling' and 'translation" to compute the normal of tl
        return p.Normalize();
}
```

Note: If the ray has two intersection points, return the closer positive value. You can ignore cases where t<0.

Note: The following variables and functions are already defined for you to use:

Vector: v1+v2 - adds the vectors v1 and v2 and returns resulting vector

Vector: v1-v2 - subtracts v2 from v1 and returns resulting vector

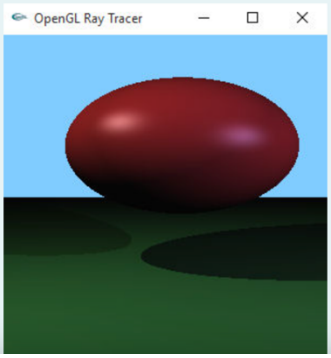
Vector: v1/v2 - returns the component-wise division of the vectors (v1.x/v2.x, v1.y/v2.y, v1.z/v2.z)

Vector: v1 * t - scales the vector v1 by t and returns resulting vector

Vector v.Scale(float a, float b, float c) - scales each component of the vector v

float v1.Dot(Vector v2) - returns the dot product of vector v1 and v2

Please paste the completed code fragment into the answer space. If your solution is correct you should get the following image:



Theory Exam Questions

Exam Question 1: Ray tracing concept

Which of the following statements about ray tracing is *false*?

Select one:

- a. Rays can be traced through many reflections and refractions.
- b. Ray tracing can be sped up by using bounding hierarchies and/or space subdivision techniques.
- c. Tracing rays backwards from the eye is much faster than tracing them from the light sources.
- d. Tracing rays from the light sources is physically more accurate than just tracing them from the eye.
- e. Ray tracing is good for tracing light rays but cannot easily determine shadows.

Exam Question 2: Ray Cut-Cylinder intersection

Given is a hollow cylinder with radius r=1 and height h=3, which is aligned with the positive y-axis and the centre of the bottom circular cross section is the origin.

This cylinder is cut with a plane $\mathbf{n}.\mathbf{p}=a$, where a=1 and $\mathbf{n}=(1/6, 1/3, 1/6)^T$.

The resulting object is a "cut-cylinder" which consists of all points which lie on the original cylinder and are inside the plane, i.e., on the opposite side the plane normal is pointing to.

Given is a ray

$$p(t) = \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} + t \begin{pmatrix} -1 \\ -2 \\ -1 \end{pmatrix}$$

What is the first intersection point, if any, of the ray with the "cut-cylinder"?

Select one:

- a. $\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$
- b. $\begin{pmatrix} 1/\sqrt{2} \\ 0.5 \\ 1/\sqrt{2} \end{pmatrix}$
- c. The ray does not intersect the "cut-cylinder".
- d. $\begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}$
- e. $\begin{pmatrix} 1.5 \\ 2 \\ 0.5 \end{pmatrix}$

Exam Question 3: Ray-Plane intersection

Given is a plane 3x+2y-z=3 and a ray

$$p(t) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + t * \begin{pmatrix} -1 \\ c \\ 0 \end{pmatrix}.$$

For what value of c is the ray parallel to the plane?

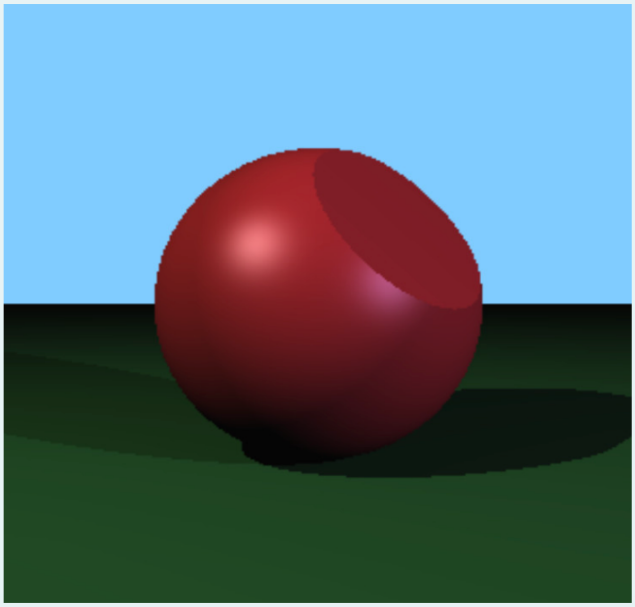
Select one:

- a. c=0.5
- b. c=-0.5
- c. c=1
- d. c=0
- e. c=1.5

Exam Question 4: Ray Cut-Sphere intersection

A plane **n**.**p**=d defines a half-space, where the half space are all points where **n**.**p**<=d.
For example, for the plane x=0 the half space are all points with an x-coordinate <=0.

Using this definition we can now define a "cut-sphere" as the intersection of a sphere S and a half-space defined by the plane P. I.e., the "cut-sphere" contains all points which are inside the sphere and inside the half-space defined by P. The image below shows an example.



Which of the statements below about the ray intersection with a "cut-sphere" (defined by a sphere S and a plane P) is *false*?

NOTE: You can assume that the "cut-sphere" is in front of the eye point.

Select one:

- a. The surface normal of a cut-sphere is either the normal of the sphere S or the normal of the plane P dependent on where the ray intersects the "cut-sphere" first.
- b. If the dot product of the ray direction and the normal of P is negative AND the ray intersects the plane P and the sphere S AND the intersection parameter with the plane P is between the two intersection parameters with the sphere S, then the intersection point with the "cut-sphere" is equal to the first intersection point with the sphere S.
- c. If the dot product of the ray direction and the normal of P is positive AND the ray intersects the plane P after it intersects the sphere S, then the intersection point with the "cut-sphere" is given by the intersection point with the sphere S.
- d. If the dot product of the ray direction and the normal of P is positive AND the ray intersects the plane P before it intersects the sphere S, then the "cut-sphere" object does not exist (i.e., the intersection of the sphere and half-space is empty).
- e. If the dot product of the ray direction and the normal of P is negative AND the ray intersects the plane P before it intersects the sphere S, then the "cut-sphere" object is equal to the sphere S (i.e., the intersection of the sphere and half-space is the sphere and nothing gets cut off.).

Programming Exam Questions

Exam Question 5: Ray-SphereWithSphereHole Intersection

Please complete the function below defining a **Ray-SphereWithSphereHole Intersection**: the intersection of a ray with a unit sphere centered at the origin where part of the unit sphere has been removed by a second sphere (i.e., the set difference of two spheres):

```
double SphereWithSphereHole::Intersect(Vector source, Vector d)
```

NOTE: The pre-loaded answer box contains already code for the ray unit sphere intersection and sorts the intersection points such that t1 is the first intersection point with the original sphere.
Add additional code such that the function returns instead the intersection points with the "SphereWithSphereHole".
The second sphere (which is removed from the sphere object) is defined by the variables **centre** (type Vector) and *radius* (type float).
Think about how you can compute the ray intersection with a sphere with the given centre and radius - it can be done by using the same code as already given with just two variables changes.

HINT: Compute the ray intersection with the second sphere and develop an algorithm to decide where the ray intersects the "SphereWithSphereHole".

IMPORTANT: If the ray first intersects the surface formed by removing the second sphere from the sphere object, i.e., the intersection point is inside the original sphere, then you need to set **sphericalHoleIntersected = true;**

This is used in the normal calculation and without the automarker will mark your solution as false.

Note: The following variables and functions are already defined for you to use:

The sphere representing the hole defined by the variables **centre** (type Vector) and *radius* (type double).

Vector: v1+v2 - adds the vectors v1 and v2 and returns resulting vector

Vector: v1-v2 - subtracts v2 from v1 and returns resulting vector

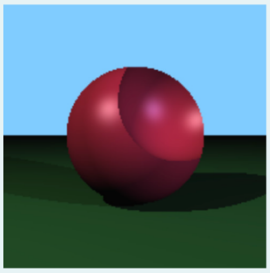
Vector: v1 * t - scales the vector v1 by t and returns resulting vector

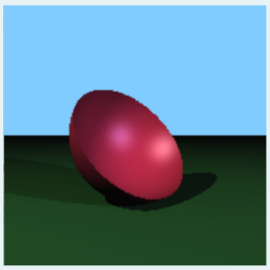
Vector v.Scale(float a, float b, float c) - scales each component of the vector v

float v1.Dot(Vector v2) - returns the dot product of vector v1 and v2

Vector v1.Cross(Vector v2) - returns the cross product of vector v1 and v2

After completing the code in the pre-loaded answer box you should get the images below:





Provided Structure code and hints:

```
double SphereWithSphereHole::Intersect(Vector source, Vector d)
{
    float t; // the parameter t for the closest intersection point or ray with
             // the "sphere with sphere hole". If no intersection t=-1.0

    // A, B, and C are the parameters of quadratic equation for finding the
    // ray intersection parameter t (see slide 16 of the "Ray Tracing" lecture notes)
    float A = d.Dot(d);
    float B = 2 * source.Dot(d);
    float C = source.Dot(source) - 1;

    // BEGIN SOLUTION RAY-SPHERE INTERSECTION
    if (B * B - 4 * A * C <= 0) return t=-1;  // no hit

    float t1;
    if (B > 0)   // for numerical precision
        t1 = (-B - sqrt(B * B - 4 * A * C)) / (2 * A);
    else
        t1 = (-B + sqrt(B * B - 4 * A * C)) / (2 * A);

    float t2 = C / (A * t1); // easier way to get t2
    // END SOLUTION RAY-SPHERE INTERSECTION


    // if t1>t2 swap, so t1 is always first intersection point
    float temp;
    if (t1 > t2) {
        temp = t1;
        t1 = t2;
        t2 = temp;
    }
    t = t1;

    sphericalHoleIntersected = false; // variable of type bool - true if ray first
                                      // intersects the flat part of the cut sphere

    // ================================================================
    // == The above method code computes the intersection parameters of ==
    // == the ray with the sphere. Add additional code such that the    ==
    // == function returns instead the intersections points with the    ==
    // == "sphere with sphere hole".                                    ==
    // == Tip: Compute the intersection with the sphere representing    ==
    // ==      the hole and based on the intersection parameters decide ==
    // ==      where the "Sphere with sphere hole" was intersected.     ==
    // == IMPORTANT: If the ray first intersects the surface formed by  ==
    // ==            removing the second sphere from the sphere object, ==
    // ==            i.e., the intersection point is inside the         ==
    // ==            original sphere, then you need to set              ==
    // ==            sphericalHoleIntersected = true;                   ==
    // ================================================================
    // INTERSECTION OF RAY WITH THE SPHERE REPRESENTING THE HOLE
    // The second sphere removed from the sphere object is defined by the variables
    // Vector centre // centre of removed sphere
    // float radius  // radius of removed sphere

    // Please complete the missing code

    return t;
}
```

Exam Question 6: Normal of SphereWithSphereHole

Please implement the function for computing the normal at a point **p** of a unit sphere centered at the origin with part of it removed by a second sphere defined by the variables **centre** (type Vector) and *radius* (type float) (i.e., the normal of the set difference of two spheres). The function should have the following signature:

```
Vector SphereWithSphereHole::Normal(Vector p)
```

NOTE: The following variables are defined:

Vector p - the point on the surface for which we want to compute the normal.

Vector c - the centre of the second sphere (which removes part of the unit sphere)

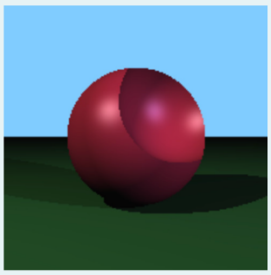
float radius - the radius of the second sphere (which removes part of the unit sphere)

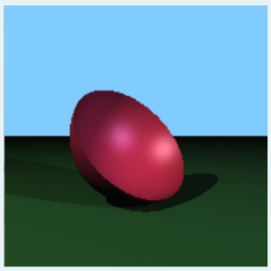
bool sphericalHoleIntersected - the variable is *false* if p lies on the unit sphere and *true* if p lies on the part of the surface created by the second sphere.

Note: The following functions are already defined for you to use:

Vector: v1+v2 - adds the vectors v1 and v2 and returns resulting vector
Vector: v1-v2 - subtracts v2 from v1 and returns resulting vector
Vector: v1 * t - scales the vector v1 by t and returns resulting vector
Vector v.Normalize() - returns the normalised version of the vector v
float v1.Dot(Vector v2) - returns the dot product of vector v1 and v2
Vector v1.Cross(Vector v2) - returns the cross product of vector v1 and v2

After completing the code you should get the images below:





Exam Question 7: Ray-CutSphere Intersection

In this exercise you need to complete the function below defining a **Ray-CutSphere Intersection**:

```
double CutSphere::Intersect(Vector source, Vector d)
```

NOTE: The pre-loaded answer box contains already code for the ray sphere intersection and sorts the intersection points such that t1 is the first intersection point with the original sphere.
Add additional code such that the function returns instead the intersection points with the "cut-sphere".
The cutting plane **n**.**p**=*a* is defined by the variables **n** (type Vector) and *a* (type double).

HINT: Compute the ray-plane intersection and develop an algorithm to decide whether the ray intersects the round (un-cut part) of the sphere, the cutting plane, or doesn't intersect the cut sphere at all (e.g. the ray might pass through the cut-off part of the sphere)

IMPORTANT: If the ray first intersects the cutting plane of the cut sphere (the flat part of it) then you need to set **cuttingPlaneIntersected = true;** . This is used in the normal calculation and without the automarker will mark your solution as false.

Note: The following variables and functions are already defined for you to use:

The plane **n**.**p**=*a* is defined by the variables **n** (type Vector) and *a* (type double).

Vector: v1+v2 - adds the vectors v1 and v2 and returns resulting vector

Vector: v1-v2 - subtracts v2 from v1 and returns resulting vector

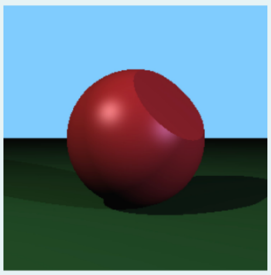
Vector: v1 * t - scales the vector v1 by t and returns resulting vector

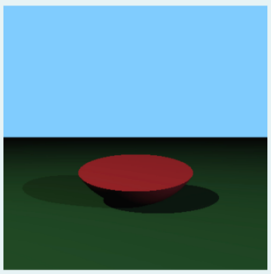
Vector v.Scale(float a, float b, float c) - scales each component of the vector v

float v1.Dot(Vector v2) - returns the dot product of vector v1 and v2

Vector v1.Cross(Vector v2) - returns the cross product of vector v1 and v2

After completing the code in the pre-loaded answer box you should get the images below:





Provided structure code and hints

```
double CutSphere::Intersect(Vector source, Vector d)
{
    // A, B, and C are the parameters of quadratic equation for finding the
    // ray intersection parameter t (see slide 16 of the "Ray Tracing" lecture notes)
    float A = d.Dot(d);
    float B = 2 * source.Dot(d);
    float C = source.Dot(source) - 1;

    float t; // the parameter t for the closest intersection point or ray with the sphere. If no intersection t=-1.0

    // BEGIN SOLUTION RAY-SPHERE INTERSECTION
    if (B * B - 4 * A * C <= 0) return t=-1;  // no hit

    float t1;
    if (B > 0)   // for numerical precision
        t1 = (-B - sqrt(B * B - 4 * A * C)) / (2 * A);
    else
        t1 = (-B + sqrt(B * B - 4 * A * C)) / (2 * A);

    float t2 = C / (A * t1); // easier way to get t2
    // END SOLUTION RAY-SPHERE INTERSECTION


    // if t1>t2 swap, so t1 is always first intersection point
    float temp;
    if (t1 > t2) {
        temp = t1;
        t1 = t2;
        t2 = temp;
    }
    t = t1;

    cuttingPlaneIntersected = false; // variable of type bool - true if ray first intersects the flat part of the cut sphere
    // ================================================================
    // == The above method code computes the intersection parameters of ==
    // == the ray with the sphere. Add additional code such that the    ==
    // == function returns instead the intersections points with the    ==
    // == "cut-sphere".                                                 ==
    // == Tip: Compute the ray-plane interscetion and develop an        ==
    // ==      algorithm to decide whether the ray intersected the      ==
    // ==      round (un-cut part) of the sphere, the cutting plane,    ==
    // ==      or doesn't intersect the cut spere at all (e.g. the ray  ==
    // ==      might pass through the cut-off part of the sphere        ==
    // == IMPORTANT: If the ray first intersects the cutting plane of   ==
    // ==            the cut sphere (the flat part of it) then you need ==
    // ==            to set cuttingPlaneIntersected = true;             ==
    // ==            This is used in the normal calculation and without ==
    // ==            the automarker will mark your solution as false    ==
    // ================================================================

    // Please complete the missing code

    return t;
}
```

Exam Question 8: Normal of CutSphere

In this exercise you need to complete the normal calculation for the **Ray-CutSphere Intersection** from the previous question. Note that you can complete this question even if you were unable to implement the intersection function correctly.

Please implement the function

`Vector CutSphere::Normal(Vector p)`

NOTE: The function should return the correct normal of the "cut: unit sphere at the ray intersection point **p**. Note that the cut-sphere is a unit sphere with centre at the origin. The normal of the cutting plane is **n** (type Vector).

NOTE 2: You can assume that the boolean variable *cuttingPlaneIntersected* is defined and that it is *true* if **p** lies on the flat part of the "cut" unit sphere and *false* otherwise.

Note: The following variables and functions are already defined for you to use:

The plane **n**.**p**=*a* is defined by the variables **n** (type Vector) and *a* (type double).

Vector: v1+v2 - adds the vectors v1 and v2 and returns resulting vector

Vector: v1-v2 - subtracts v2 from v1 and returns resulting vector

Vector: v1 * t - scales the vector v1 by t and returns resulting vector

Vector v.Scale(float a, float b, float c) - scales each component of the vector v

float v1.Dot(Vector v2) - returns the dot product of vector v1 and v2

Vector v1.Cross(Vector v2) - returns the cross product of vector v1 and v2

Vector v.Normalize() - returns the normalised version of the vector v

After completing the code in the pre-loaded answer box you should get the image below: