CS535 Computer Graphics
Homework Assignment 7 Solution Set (40 points)

Due date: 12/02/2023

---

1. A modern CPU can have 4 or 8 cores, but a modern GPU can have thousands. These GPU cores can be used for computationally intensive tasks through the use of ***compute shaders***. Compute shaders are programmed in GLSL and run independently. A computer shader can perform parallel computing in the following sense: if a compute shader is required to perform a task on *n* different data sets, one can first creates n copies of the compute shader (invokes the compute shader n times) and then assign each copy of the compute shader a different data set (assign a different task ID (invocationID)). These copies of the compute shader then run in parallel to perform the task on assigned data sets. In the following box, explain how these two things are implemented in a computer shader program, especially the GLSL commands/variables needed for these two steps. (10 points)

First, one needs to use the following command to define a 1D, 2D or 3D grid of *n* nodes with each node being a work group (core, but simply think of it as a copy of the compute shader program):

glDispatchCompute( )

Here we assume the size of each work group to be 1.

Next, in the main() of the compute shader program, each work group will get a task ID assigned by the special variable ***gl_GlobalInvocationID.**** and then perform the assigned task independently.

2. Given a 2D box defined by four bounding edges

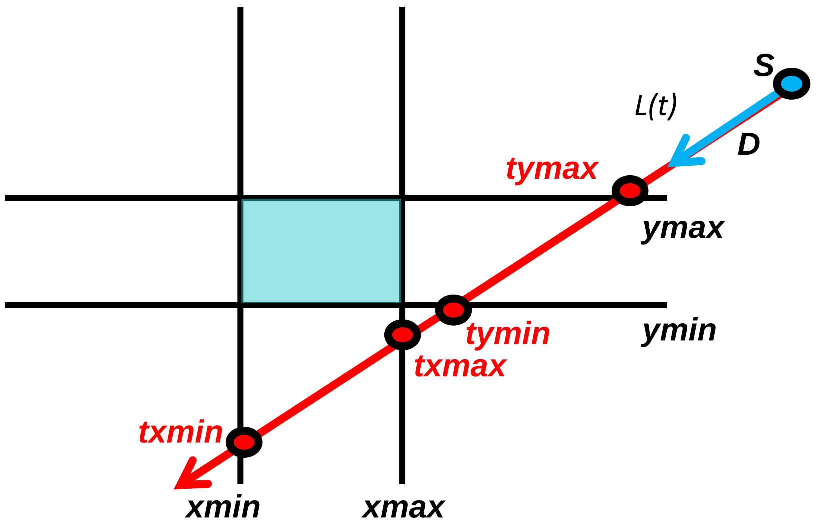
   $x = xmin \qquad x = xmax \qquad y = ymin \qquad y = ymax$

   and a ray defined by

   L(t) = S + tD

   where S=(a,b) is the start point and D=(c,d) is the (normalized) vector of direction (see the following figure), let $t_{xmin},$ $t_{xmax},$ t_{ymin} and t_{ymax} be the parameters of the intersection points of the ray with the bounding edges $x =$

$xmin$, $x = xmax$, $y = ymin$ and $y = ymax$, respectively. If we use the notations defined in slide 90 of the notes “Ray Tracing I”, what would t_{near} and t_{far} be? Show your work in the following red box. (5 points)



txmin < txmax < tymin < tymax

First note that, in this case, *txmin > txmax > tymin > tymax.*

Hence, we have

$t_{xminDist}$=min(t_{xmin}, t_{xmax}) = t_{xmax} $t_{xmaxDist}$=max(t_{xmin}, t_{xmax}) = t_{xmin}

$t_{yminDist}$=min(t_{ymin}, t_{ymax}) = t_{ymax} $t_{ymaxDist}$=max(t_{ymin}, t_{ymax}) = t_{ymin}

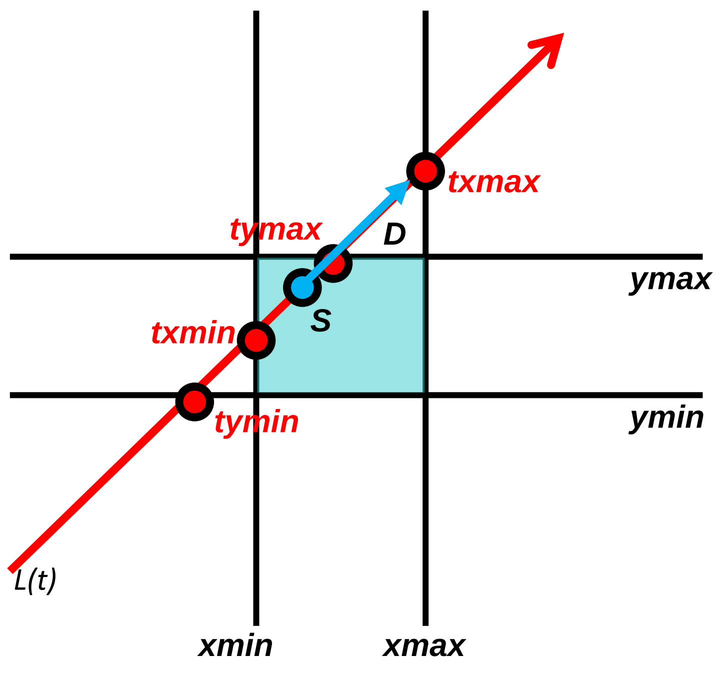
Consequently,

t_{near}=max($t_{xminDist}$, $t_{yminDist}$) = max(t_{xmax}, t_{ymax}) = t_{xmax}

t_{far}=min($t_{xmaxDist}$, $t_{ymaxDist}$) = min(t_{xmin}, t_{ymin}) = t_{ymin}

Since $t_{near} > t_{far}$ so the ray didn’t intersect the box.

3. If the case shown in the following figure is given, then what would t_{near} and t_{far} be? Show your work in the following red box. (5 points)



tymin < txmin < tymax < txmax

In this case we have tymin < txmin < tymax < txmax

Hence,

$t_{xminDist}$=min(t_{xmin}, t_{xmax}) = t_{xmin} $t_{xmaxDist}$=max(t_{xmin}, t_{xmax}) = t_{xmax}

$t_{yminDist}$=min(t_{ymin}, t_{ymax}) = t_{ymin} $t_{ymaxDist}$=max(t_{ymin}, t_{ymax}) = t_{ymax}

Consequently,

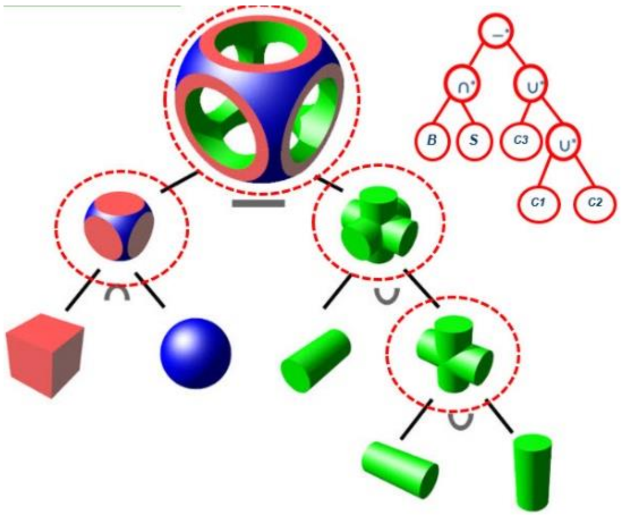
t_{near}=max($t_{xminDist}$, $t_{yminDist}$) = max(t_{xmin}, t_{ymin}) = t_{xmin} < 0

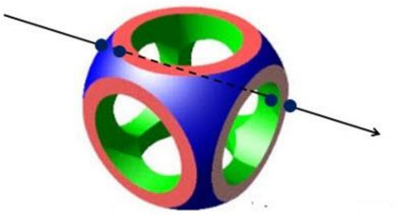
t_{far}=min($t_{xmaxDist}$, $t_{ymaxDist}$) = min(t_{xmax}, t_{ymax}) = t_{ymax}

Since $t_{near} < t_{far}$ and $t_{far} > 0$ so the ray intersects the box.

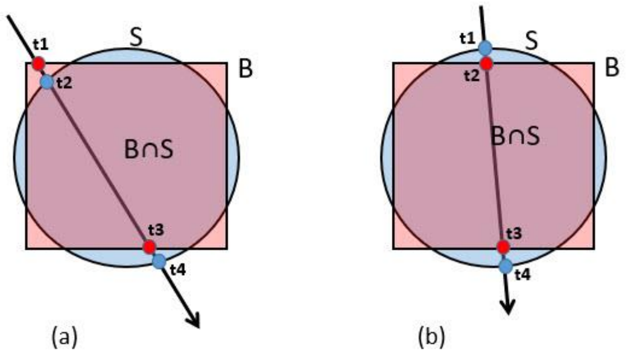
4. [Solid Modeling] Given a virtual object represented as a CSG (Constructive Solid Geometry) tree (see Section 10.10 for the definition of a CSG tree), one can use ray casting or even ray tracing technique to render this virtual object on screen. To use ray casting technique to render a CSG object, we need to find the intersection points of each ray with the object. For instance, for the CSG object given below (left figure), for the given ray, we need to

find the parameters of the intersection points of the ray with the object (right figure).





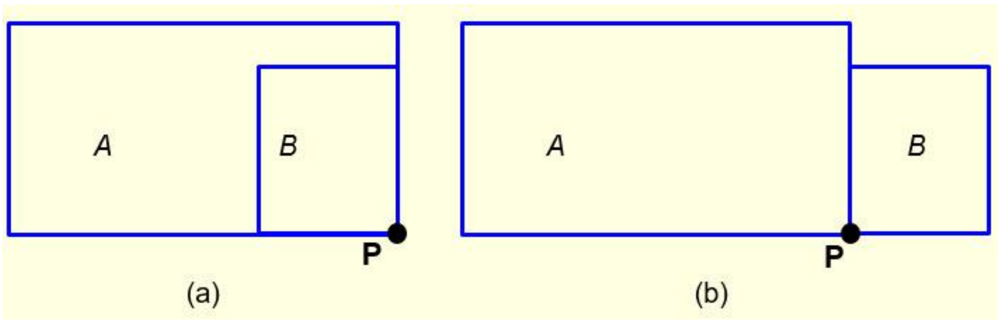
In the following, use the two given cases (intersection of a 2D sphere and a 2D cube, such as the intersection of B and S in the above CSG representation) to explain which two parameters should be reported for each case and why. (10 points)



In case (a), by taking the **intersection** of intervals [t1, t3] (the portion of the ray that is inside B) and [t2, t4] (the portion of the ray that is inside S), we get the interval [t2, t3] (note that t1 < t2 < t3 < t4). So the intersection points of the ray with B∩S in this case are t2 and t3.

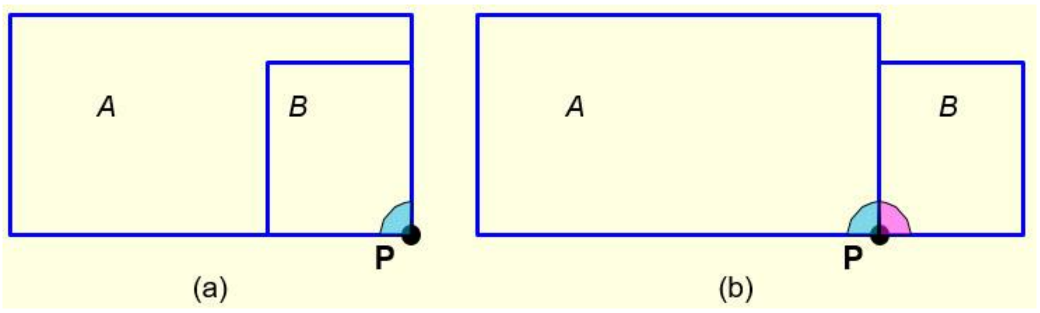
In case (b), by taking the **intersection** of intervals [t1, t4] (the portion of the ray that is inside S) and [t2, t3] (the portion of the ray that is inside B), we get the interval [t2, t3] (note that t1 < t2 < t3 < t4). So the intersection points of the ray with B∩S in this case are also t2 and t3.

5. [Solid Modeling] The lookup tables in slide 23 of the notes "Ray Tracing II" do not work for case (a) and case (b) shown below. This does not mean the technique is not valid, only that a few things of the lookup tables have to be modified. Show me how these lookup tables should be modified so that the technique would work for the following cases as well. (5 points)

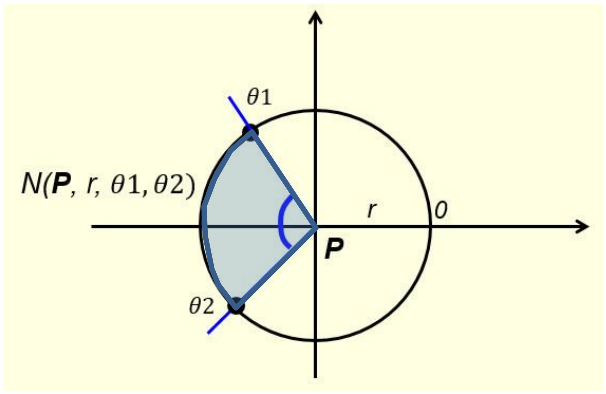


Solution:

Note that in the cases shown in the above figure, the regular neighborhood of **P** in either A or B is no longer a half circle, but a quarter circle as shown in the following figure.



Actually, in some cases, we could even have a regular neighborhood smaller than a quarter circle. A better way to define a regular neighborhood should be in terms of degree instead of size. We shall use N(P, r, $\theta 1, \theta 2$) to denote a regular neighborhood for a point **P** with radius r and starting angle $\theta 1$ and ending angle $\theta 2$ as follows:



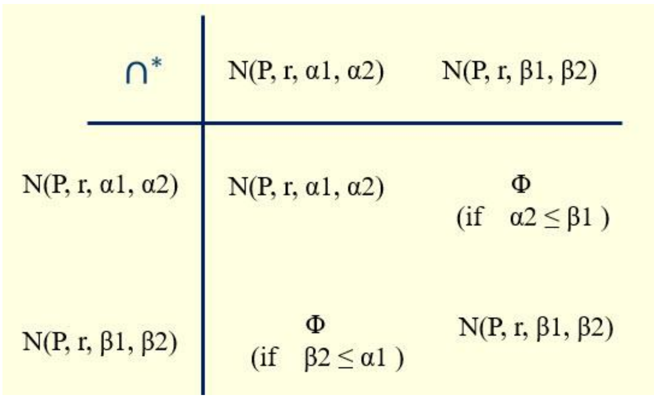
Using this notation, the regular neighborhood of the point P in (a) would be

N(P, r, 90^o , 180^o)

and in case (b) the regular neighborhood of P in A would be the same, but the regular neighborhood of P in B would then be

N(P, r, 0^o, 90^o)

If we use N(P, r, $\alpha1, \alpha2$) to denote a regular neighborhood of P in A and N(P, r , $\beta1, \beta2$) to denote a regular neighborhood of P in B, then a lookup table for intersection on A and B should be defined as follows:



$\cap^*$	N(P, r, α1, α2)	N(P, r, β1, β2)
N(P, r, α1, α2)	N(P, r, α1, α2)	Φ (if $\alpha2 \le \beta1$)
N(P, r, β1, β2)	Φ (if $\beta2 \le \alpha1$)	N(P, r, β1, β2)

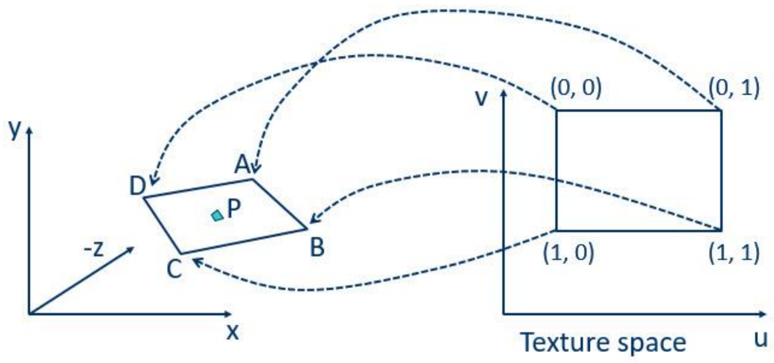
Lookup tables for union and difference can be defined similarly.

6. Can texture mapping be integrated into the ray tracing process? Justify your answer and be specific. For instance, if your answer is YES, you must tell me how it is done, including how you get the texture space coordinates, how to handle out-of-range problem, and how to handle the do-not-directly-match-a-texel problem. If your answer is NO, then you have to tell me why this is not possible, with all the supporting arguments. (5 points)

Sol.
Yes, ray tracing can be used with texture mapping to reproduce surface texture on an otherwise smooth polygon surface.

How is this done? The most efficient way to do this is to perform the following steps after the intersection point to the closest object has been determined so that we don't spend time calculating texture coordinates for hidden objects.

```
getTexture(point p)
Convert p from world coordinates to local object coordinates (lp)
Convert lp point to texture space coordinates (u,v)
If the (u,v) coordinates are out of range use tiling, mirroring, or another technique to get appropriate coordinates.
If the (u,v) coordinates do not match directly use the nearest neighbor or interpolation to get the correct color
Return the color to the ray tracer
```



Since we are still just returning a color to the ray tracer, texture mapping can be easily integrated into the tracing process.