TDA362/DIT224 Computer Graphics

EXAM

(Same exam for both CTH- and GU students)

Friday, January 12th, 2024, 08:30 - 12:30

Examiner

Ulf Assarsson, tel. 031-772 1775

Permitted Technical Aids

None, except English dictionary

General Information

Numbers within parentheses states the maximum given credit points for the task. Solutions shall be clear and readable. Too complex solutions can cause reductions in the provided number of points

Questions to examiner during exam

will be possible approximately one hour after the start of the exam. If anything is unclear – remember what has been mentioned on the lectures, in the slides and course book and do your best.

Grades

In order to pass the course, passed exam + exercises (or exam + project) are required. The final grade is calculated from the exam grade. The exam is graded as follows

CTH: 24p ≤ **grade 3** < 36p ≤ **grade 4** < 48p ≤ **grade 5**

GU: 24p ≤ **G** < 45p ≤ **VG**

Max 60p

Grades are announced by the LADOK system ~3 weeks after the exam

Solutions

will be announced on the course home page.

Review

Review date (granskningsdatum) is announced on the course home page.

Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

Question 1

a) **[1p]** What does the fragment shader do? (Hint: what is its input and output and when is it executed?)
Answer: Receives interpolated values from vertex shader and computes fragment color. E.g., refines lighting, adds textures, may modify depth.

b) **[2p]** Why do we want to use double buffering? Explain what can happen without it.
Answer: to avoid screen tearing. Also explain screen tearing - see lecture 1.

c) **[2p]** Which are the four buffers a default frame buffer typically (or often) consists of?
Answer: Front Color Buffer, Back Color Buffer, Depth Buffer, Stencil Buffer

d) **[2p]** Assume that you are creating a system where you define an object's transform from model space to world space by using a rotation matrix **R** for the object's rotation around its origin, a scaling matrix S for its scale, and a translation matrix T for its world-space positioning. For a vertex **v** of the object, show how you compute the transformed vertex **v'**. (Use the recommended order of the matrices. A one-line answer **v'**=... is enough.)
Answer: v' = (**TRS**) **v**

e) **[3p]** State which matrix components control 1) **scaling** in **x-**, **y-**, and **z**-direction, 2) **rotations** in general, and 3) **translation** in the **x-**, **y-**, and **z**-direction?

$$\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$$

Answer:
rotation: abcefgijk.
scaling: x: a, y: f, z: k
translation: x: m, y: n, z: o

Question 2

a) **[1p]** While a normal might be normalized in the vertex shader, why do we generally need to normalize it again in the fragment shader?
Answer: As values get interpolated between the vertices from the vertex shader to the fragment shader, the interpolated normal in not necessarily normalized.

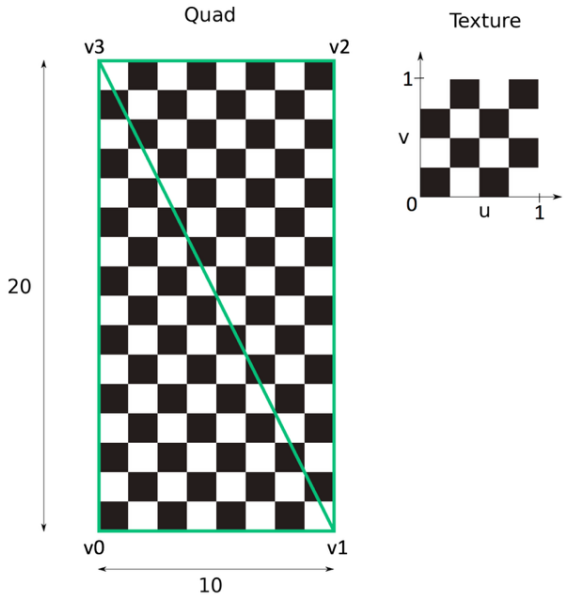
b) **[1p]** Is an alpha channel in the color buffer needed to render transparent objects correctly? Motivate.
Answer: No. The standard transparency-blend equation only uses the src-alpha and not dest-alpha (see OpenGL).

c) **[1p]** What is the difference between supersampling and multisampling?
Answer: multisampling only runs the fragment shader once per fragment (not once per sample-inside-each-pixel). I.e., multisampling shares computations, e.g. executes fragment shader for only one sample but takes several depth samples.

d) **[1p]** Up to how many texel accesses are required for highest-quality 16x anisotropic filtering of one texture-lookup request? And why?
Answer: 8x16 = 128.
I.e., 8 texel accesses per trilinear-filtered lookup, and up to 16 texture such trilinear mipmap lookups along the line of anisotropy.

e) **[2p]** Describe how jittering works and also tell me what is the main advantage compared to other sampling schemes.
Answer: replaces undersampling artifacts with noise. Take a random sample position per subcell.

f) **[2p]** Assume you want to texture a quad with a repeating chessboard pattern (see the figure). You start by setting the texture wrap option to REPEAT. What texture coordinates (*u,v*) should you specify for each vertex v0,v1,v2,v4 to achieve the desired effect?
Answer: v0:(0,0), v1:(2,0), v2: (2,4), v3:(0,4)



g) **[2p]** Assume that you are rendering an impostor (i.e, billboard) as a rectangular quad. The impostor's texture represents a tree and contains lots of fully transparent texels. How do you assure that objects that are later (for the same frame) rasterized behind the transparent parts of the impostor will show on the screen? (**Hint:** use the fragment shader.)
Answer: alpha test, fragment kill, i.e., kill fragment if alpha = 0 in fragment shader.

Question 3

a) **[2p]** Assume that you have enabled backface culling. How does the hardware determine if a triangle will be backfacing or frontfacing? Draw an explanatory image!
Answer: The vertices v0, v1, v2, are projected onto the screen and if they appear in anti-clockwise order (right-hand side rule), the triangle is (by default) assumed to be front facing. Otherwise, backfacing. (The order could be reversed by specifying CW-order.)

b) **[2p]** To draw transparent objects using for instance OpenGL, you typically divide the triangles in two groups: the transparent ones and the opaque ones. 1) Which of these two groups needs to be sorted, 2) why, 3) and in which order?
Answer: 1) the transparent triangles, 2) for correct blending, 3) back-to-front.

c) **[1p]** Normally, you would want to draw all geometry that is in front of the camera. So, why is a near and far plane used for the view frustum?
Answer: Near plane is used to avoid a degenerate projection plane and get better z-precission in the z buffer. Far plane also increases z-precision but only slightly.

d) **[2p]** Show how to compute the intersection (in 3D) between a ray and an infinite cylinder centered on the z-axis.
Answer:

1) Equation for cylinder: $||x^2 + y^2|| = r$, or simply in 2D by dropping the z components: $||\mathbf{p}-\mathbf{c}||=r \rightarrow$ [square and note that $\mathbf{c}=(0,0)$] $\rightarrow \mathbf{p}\cdot\mathbf{p} = r^2$.
2) Ray function: $\mathbf{r}(t)=\mathbf{o}+t\mathbf{d}$.
3) Replace $\mathbf{p}$ by $\mathbf{r}(t)$: gives $(o_x + td_x)^2 + (o_y + td_y)^2 = r^2$, or simply in 2D: $(\mathbf{o}+t\mathbf{d})\cdot(\mathbf{o}+t\mathbf{d}) = r^2$.
4) Solve for *t* and consider that there are two solutions.
E.g.: $(\mathbf{o}\cdot\mathbf{o}) + 2t(\mathbf{o}\cdot\mathbf{d}) + t^2(\mathbf{d}\cdot\mathbf{d}) = r^2$
$(\mathbf{d}\cdot\mathbf{d})t^2 + 2t(\mathbf{o}\cdot\mathbf{d}) + (\mathbf{o}\cdot\mathbf{o}) - r^2 = 0$. This is a standard scalar quadratic equation for *t*.
$(\mathbf{d}\cdot\mathbf{d})t = -(\mathbf{o}\cdot\mathbf{d}) \pm \text{sqrt}((\mathbf{o}\cdot\mathbf{d})^2 - (\mathbf{o}\cdot\mathbf{o}))$. You may assume $||\mathbf{d}|| = 1$.
5) Also, indicate that solution is the 3D point $\mathbf{o} + t\mathbf{d}$.

e) **[3p]** Explain an efficient method for determining intersection between a ray and a box (in 3D).
Answer: Find t_{min} and t_{max} for the intersections against each of the three slabs. Keep *max* of the three t_{min} and *min* of the three t_{max}. If $t_{min} < t_{max}$, there is an intersection. (Check for degenerate case when ray parallel to slab.)

Question 4

a) **[3p]** For some scene, draw:
- an Axis-Aligned Bounding Box hierarchy
- an OBB hierarchy
- an Octree or Quadtree
- an Axis-Aligned Binary Space-Partitioning Tree
- a grid
- a recursive and a hierarchical grid

You can do all your drawings in 2D for simplicity and you may use different scenes for the different hierarchies. (You don't need to show the corresponding tree structures). The **differences** in the spatial divisions between all the types of data structures **must be clear** from your examples, for any score.
Answer:

b) **[1p]** Where in the tree are the triangles stored for an Axis-aligned BSP-tree?
Answer: in the leaves.

c) **[1p]** Why is bubble sort efficient when we have high frame-to-frame coherency regarding the objects to be sorted per frame?
Answer: Bubble-sort (or insertion sort) has expected runtime of resorting already almost sorted input in O(n) instead of O(n log n), where n is number of elements.

d) **[1p]** How do you nicely terminate recursion in ray tracing if you do not simply want to terminate at a maximum recursion depth?
Answer: Terminate when the ray's influence on the final pixel color is below a certain threshold. (Send a weight with the reflection/refraction rays.)

e) **[4p]** Describe the structure of a typical ray tracer by using the functions main(), trace(), shade(), and findClosestIntersection(). I.e., describe what these functions do and which functions they call (who calls who).
Answer: main()-calls trace() for each pixel.
Trace(): returns the color of the closest intersected point. Calls findClosestIntersection() and then calls shade() for the point.
Shade(): computes color. For each light, shoot shadow ray. If in shadow -> compute diff/spec light. Always compute amb light. Calls trace recursively for reflection/refraction ray.

Question 5

a) **[3p]** This is the rendering equation. Explain the equation and all of its components. For points, include an explanatory figure in your answer.

$$L_o = L_e + \int_{\Omega} f_r(\mathbf{x}, \omega, \omega') L_i(\mathbf{x}, \omega')(\omega' \cdot \mathbf{n}) d\omega'$$

Answer: f_r is the BRDF, ω' is incoming direction, **n** is normal at point **x**, Ω is hemisphere around **x** and **n**, L_i is incoming radiance, **x** is position on surface, ω is outgoing direction vector.
$L_o(\mathbf{x},\omega)=L_e(\mathbf{x}, \omega)+L_r(\mathbf{x}, \omega)$ (slightly different terminology than Kajiya). I.e., outgoing radiance =emitted + reflected radiance. Integral represents reflected radiance.

b) **[2p]** Describe the Fresnell effect for dielectric materials and metals, respectively. (You may draw graphs or describe with words.)
Answer: dielectrics: Wavelength independent (mostly). High transmittance/low reflectivity for low angles. Low transmittance/high reflectivity for high angles.
Metals: Wavelength dependent and typically high reflectivity for all angles (there can be a slight dip e.g. around 85 degrees)

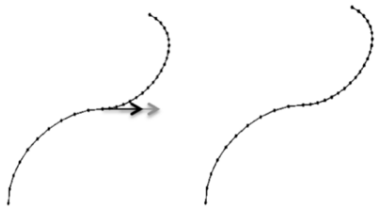
c) **[3p]** Describe the shadow map algorithm.
Answer: 1. Render a shadow (depth) map from the light source.
2. Render image from the eye. For each generated pixel, transform/warp the x,y,z-coordinate to light space and compare the depth with the stored depth value in the shadow map (at the pixel position (x,y).
If greater → point is in shadow.
Else → point is not in shadow.
(Bias/offset is necessary due to disretization and precision problems.)

d) **[1p]** How do you modify the z-pass algorithm into the z-fail algorithm?
Answer: The problem with the eye located within shadow volumes (stencil count gets wrong) is solved. Counting is done to infinity instead of to the eye. Invert the depth test and stencil inc/dec. I.e., use depth test GREATER and inc for backfacing shadow quads instead of vice versa. Also render near- and far capping shadow volume planes.

e) **[1p]** Assume you have an **incoming** light ray **l** at a surface point **p** with the normal **n** (which is normalized). Compute the reflection vector **r**. Draw figure!
Answer: r= **l** - 2(**n**•**l**)**n**.

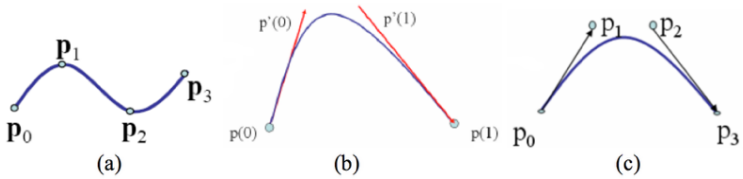
Question 6

a) **[1p]** What continuity does the curve below have? Motivate. (I.e., the two curves are exactly the same curve, but with and without arrows to depict the type of gradient discontinuity).



Answer: G^1 continuity. Different gradients, but same direction.

b) **[3p]** Tell which type of curve each image corresponds to. You can choose between Bezier curve, Interpolation-curve, and Hermite-curve. To get any points, you must also motivate your choice!



Answer: a) interpolation curve since the curve goes through the control points. b) Hermite-curve since gradients are specified per control point. c) Bezier-curve since two intermediate points are used to approximate the gradients in the end points.

c) **[1p]** Assume **p**=(7,5,2,9). Perform the homogenisation step on **p**.
Answer:

d) **[1p]** Manually normalize the vector **x**=(1,0,3).
Answer:

e) **[2p]** Compute how far above or below point **p**=(3,-1,-3) is with respect to the plane with **n**=(4,2,-4)/sqrt(36) and d=-3.
Answer: Signed distance = n • p + d = 22 / sqrt(36) + -3.

f) **[2p]** There are 4 main taxonomies of hardware, based on where in the GPU-pipeline sorting in screen space is done. Which are these four? (Names are enough. You do not need to describe them if you don't want to.)
Answer: Sort-first: sort the triangles spatially before the geometry stage.
Sort-middle: sort the triangles spatially after the geometry stage.
Sort last fragment: sort the rasterized fragments spatially.
Sort last image: Have a frame buffer per pipeline. Merge the frame buffers after full rendering.
(See lecture 13.)