TDA362/DIT223 Computer Graphics
EXAM

(Same exam for both CTH- and GU students)

Saturday, January 13th, 2018, 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.

Question 1

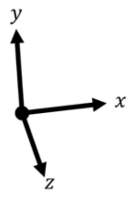
a) **[2p]** Explain what the z-buffer is used for and how it works.
Answer: Use a buffer called the z or depth buffer to store the depth of the closest object at each pixel found so far. As we render each polygon, compare the depth of each pixel to depth in z buffer. If less, place shade of pixel in color buffer and update z buffer.

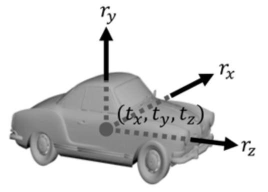
b) **[1p]** What is the advantage of Bresenham's line drawing algorithm?
Answer: uses only integers

c) **[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.

d) **[5p]** Consider a game where the model-matrix (the position and orientation) for a car is described by a translation matrix, *T*, and a rotation matrix, *R*, as given below.

$$M = TR = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} & 0 \\ r_{xy} & r_{yy} & r_{zy} & 0 \\ r_{xz} & r_{yz} & r_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$





1. (1p) When the user presses a key, the car should move 2.0 units in the world-space *x* direction. How would you modify the translation matrix?
2. (2p) How would you modify the rotation matrix for the car to turn slightly to the left?
3. (1p) When the user presses the "up-arrow" key, the car shall move forward in the direction it is facing. How do you modify the matrices?
4. (1p) We now want a camera from inside the car (at the car's position and facing in the r_z direction). How can you calculate this matrix?

Answer:

1. $t_x = t_x + 2$
2. You must show that you understand that $\boldsymbol{r_z}$ and $\boldsymbol{r_x}$ should be rotated around $\boldsymbol{r_y}$
   (E.g., from the labs):
   $\boldsymbol{r_z}$ += 0.01 $\boldsymbol{r_x}$
   normalize $\boldsymbol{r_z}$
   $\boldsymbol{r_x} = \boldsymbol{r_z} \times \boldsymbol{r_y}$
3. (t_x, t_y, t_z) += 0.1 $\boldsymbol{r_z}$
   (0.1 is just an example)
4. $V = (TR)^{-1}$

Question 2

a) **[3p]** Describe flat shading, Gouraud shading and Phong shading.
Answer: Flat shading: illumination computed using the triangle-plane's normal.
Gouraud-shading: illumination computed per triangle vertex and for each pixel, the color is interpolated.
Phong Shading: For each pixel, the normal is interpolated from the 3 vertex-normals. Full illumination is done with this interpolated normal.

b) **[3p]** How often are the vertex shader, geometry shader and fragment shader executed when one triangle is sent as input for drawing by the hardware? Explain your answer.
Answer: Three, once, and once per non-occluded fragment (unless supersampling schemes are used, which executes the fragment shader per fragment sample).

c) **[2p]** How much extra memory does it take to store a mipmap-hierarchy of a texture, compared to just the base texture itself?
Answer: ~1/3 extra (or ~33%).

d) **[2p]** What is the difference between bump mapping and displacement mapping?
Answer: Bump mapping only computes lighting as if the surface is bumpy, while displacement mapping actually (geometrically) displaces the surface according to the map.

Question 3

a) **[2p]** Assume that you have enabled backface culling. How does the hardware determine if a triangle will be backfacing or frontfacing? (Drawing an image can be useful.)
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. Far plane can be used to get better z-precision in the z buffer.

d) **[2p]** Show how to compute the intersection between a ray and a sphere (in 3D).
Answer: See slides from lecture 6.
Sphere center: **c**, and radius r
Ray: **r**(t)=**o**+t**d**
Sphere formula: ||**p**-**c**||=r
Replace **p** by **r**(t): ||**r**(t)-**c**||=r and solve for t. Return **r**(t).

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) **[4p]** Draw a simple scene and visually divide it into
- an Axis-Aligned Bounding Box hierarchy
- a Sphere hierarchy
- an OBB hierarchy
- an Octree or Quadtree
- an Axis-Aligned Binary Space-Partitioning Tree
- a Polygon-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]** Why can't we use ray tracing to do an exact object-object collision detection test?
Answer: Only tests a discrete subset of surface points and directions. A finite set of rays may miss small intersections.

c) **[1p]** What is a shadow cache? Explain what it is good for and how it works.
Answer: pointer to previous intersected triangle (primitive) by the shadow rays. Null, if last shadow ray had no intersection. Reason: speedup, potentially avoiding tree traversal.

d) **[4p]** What is Constructive Solid Geometry and how does it work?
Answer: Boolean operations between objects. Find overlap for each object and ray. Apply the Boolean operator on each interval and use the closest interval point.

Question 5

a) **[3p]** This is the rendering equation. Explain the equation and all of its components. You can include a 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: High transmittance/low reflectivity for low angles. Low transmittance/high reflectivity for high angles.
Metals: high reflectivity for all angles (there can be a slight dip e.g. around 85 degrees)

c) **[2p]** Why do you need to use a bias in the shadow map algorithm? What is the cause of the problem?
Answer: We compare two different discretizations - one from the eye and one from the camera. (To avoid z-fighting (incorrect self shadowing) and light leaking.) The view sample can lie further from the light than the shadow map sample (due to discrete sampling).

d) **[3p]** Describe how the z-fail algorithm works.
Answer: Create shadow quads from the silhouettes of the shadow caster (as seen from light source). Capping planes are necessary. Render ambient occlusion to color buffer. Turn off updating of z-buffer and rendering to color buffer. **Invert z test**, i.e., use GL_GREATER. Render front facing (as seen from camera) shadow volume polygons to stencil buffer, **decrementing** stencil values. Render back facing shadow volume polygons to stencil buffer, **incrementing** stencil values. Render diff+spec lighting contribution to color buffer where stencil buffer = 0.

Question 6

a) **[1p]** Sketch one non-continuous curve and one C^0-continuous curve (and mark which is which).
Answer:

b) **[2p]** In which ways are NURBS more general than B-Splines?
Answer: control points can be set at non-uniform intervals and they can have different weights.

c) **[1p]** Assume **p**=(6,0,1,3). Perform the homogenisation step on **p**.
Answer:

d) **[1p]** Given two vectors ω_i and ω_o, how does one calculate the half angle vector ω_h?
Answer: ω_h = normalize($\omega_i + \omega_o$).

e) **[5p] Postprocessing:** You have a scene of a spaceship which you render with a shader called FancyShader, to a texture, OffScreenTexture. To the rendered result, you want to apply a post-processing effect that generates a gray-scale image.

1. (3p) Order all of the following steps such that you would end up with a grayscale image of the spaceship in your window:
   A. Draw Spaceship
   B. Bind GrayScaleShader
   C. Draw fullscreen quad
   D. Clear rendertarget
   E. Bind OffScreenTexture to texture unit 0
   F. Bind OffScreenTexture as rendertarget
   G. Bind Window (default framebuffer) as rendertarget
   H. Bind FancyShader
2. (1p) This is an effect that could instead be implemented directly in FancyShader. Why could there be a performance advantage of doing it as a post-processing pass (especially for more computationally heavy effects)?
3. (1p) Most/Many post-processing effects, like blur, can not be implemented in FancyShader. Why not?

Answer:
1: E.g.: F, D, H, A, G, E, B, C
Actually any of these orders work: ((F → D)/H) → A → (G/E/B) → C, where → means strict order and / means any order.
2: The performance then scales with pixels, not fragments (or geometry).
3: Need to know result of neighbouring pixels.