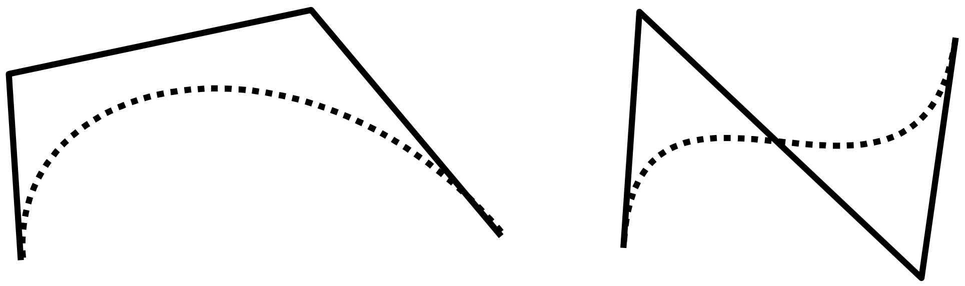
CS475/675: Computer Graphics

End-Semester Examination (Solutions), Fall 2016

This paper has **6 (six)** pages and **10 (ten)** questions, totalling **80 (eighty)** points.
No calculators, notes, books, phones, tablets, laptops etc are allowed.
(Non-red) pens only – no pencils.

1. **Bezier Curves (3 points):** Sketch the cubic bezier curves with the following control polygons. You don't need to be exact, we're looking for the general shape. In your answerbook, reproduce the control polygons and draw the curves w.r.t. them.

   Solutions:



2. **Cubic Splines (16 points):** Recall that a cubic spline can be described by the general equation

   $P(t) = at^3 + bt^2 + ct + d$

   where a, b, c, d are typically not provided directly, but are derived from user controls. Aftab wants to design his cubic splines by choosing 4 points: P_0, P_1, P_2, P_3. The curve should start at P_0, end at P_3, start with tangent $P_1 – P_0$, and end with tangent $P_3 – P_2$. Help Aftab work with these controls by answering the following questions.

   a. (4 points) Write 2 equations relating the endpoint positions to the parameters a, b, c, d, and 2 equations relating the tangents to a, b, c, d.

   Solution:

   $P_0 = d$
   $P_3 = a + b + c + d$
   $P_1 – P_0 = c$
   $P_3 – P_2 = 3a + 2b + c$

   b. (2 points) Transform the system of equations so that each equation now involves exactly one of P_0, P_1, P_2 and P_3, in the form P_i = ...

   Solution:

   $P_0 = d$
   $P_1 = c + d$

$P_2 = -2a - b + d$
$P_3 = a + b + c + d$

c. **(2 points)** Write the constraint matrix C for the splines.

Solution:

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -2 & -1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

d. **(6 points)** Solve the system of equations in (a) or (b) and write down the real-valued spline basis functions $H_i(t)$ in the following expression:

$P(t) = P_0\, H_0(t) \; + \; P_1\, H_1(t) \; + \; P_2\, H_2(t) \; + \; P_3\, H_3(t)$

Solution:

$[H_0(t) \;\; H_1(t) \;\; H_2(t) \;\; H_3(t)]^T = (C^{-1})^T\, [\, t^3 \;\; t^2 \;\; t \;\; 1]^T$, where

$$C^{-1} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -2 & 1 & 2 \\ -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Hence:

$H_0(t) = t^3 - t^2 - t + 1$
$H_1(t) = t^3 - 2t^2 + t$
$H_2(t) = -t^3 + t^2$
$H_3(t) = -t^3 + 2t^2$

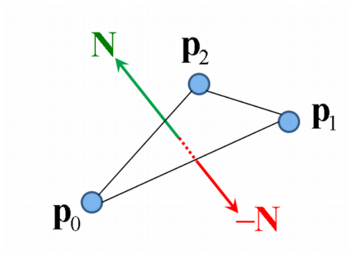
e. **(2 points)** Imagine that instead of the tangent controls, Aftab wanted the curve to pass through each of P_0, P_1, P_2, and P_3. Is the number of cubic splines satisfying this property (i) zero, (ii) one, or (iii) infinite? Briefly explain your answer.

Solution:

(iii) Infinite. The free parameters are the particular values of t at which the curve passes through P_1 and P_2. For different choices of values, different curves are obtained.

3. **Geometry (25 points):**

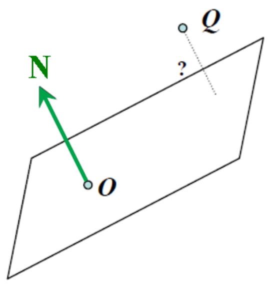
a. **(2 points)** Given the triangle of the figure below, how would you compute its unit normal $\boldsymbol{N}$? Write the formula.



Solution:

$\boldsymbol{N} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) \;/\; \| (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) \|$

b. **(3 points)** A plane passes through a 3D point $\boldsymbol{O}$ and has a unit normal $\boldsymbol{N}$, as shown in the figure below. Consider a point $\boldsymbol{Q}$ outside the plane. How would you compute the distance from $\boldsymbol{Q}$ to the plane? Write the formula. (Hint: consider using a dot product somehow.)



Solution:

Distance = $|(\boldsymbol{Q} - \boldsymbol{O}) \cdot \boldsymbol{N}|$ *(full credit even if you omit the absolute value)*

c. **(2 points)** For two curve segments to be joined together, define:

(i) C_0-continuity
(ii) C_1-continuity

Solution:

(i) C_0-continuity: The curves meet at the junction.

(ii) C_1-continuity: The curves meet and have the same first derivative at the junction. *(Ok if you write they have the same slope/tangent, but see G_1-continuity. –0.5 for omitting to write that they meet, i.e. are C_0-continuous.)*

d. **(3 points)** True or false? No explanations needed.

(i) If an object is transformed by a rotation, transforming its surface normal vectors by the same rotation will leave them perpendicular to the surface.

Solution: True, since a rotation is a rigid transform (not true in general).

(ii) The inverse of a rotation matrix is always equal to its transpose.

Solution: True. It's an orthonormal matrix.

(iii) Mapping a 2D image onto a surface in 3D always causes distortions in the resulting texture on that surface.

Solution: False. In the trivial case, the surface could be planar. But even otherwise, it's possible to map the image to any developable surface (one which can be unrolled into a plane without stretching or shrinking, e.g. a cylinder) without metric distortion (changing distances, measured on the surface, between points).

e. **(4 points)** Which of the following operations are valid? Write “Valid” or “Invalid” for each one in your answerbook.

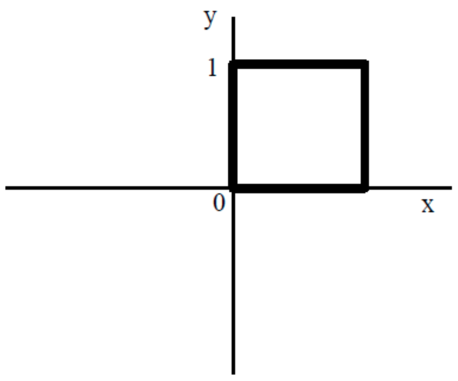
Solution:

(i) Position = Scalar * Position	**Invalid**
(ii) Direction = Position – Position	**Valid**
(iii) Position = Position + Direction	**Valid**
(iv) Direction = Direction + Direction	**Valid**
(v) Position = Position – Position	**Invalid**
(vi) Position = Position + Position	**Invalid**
(vii) Direction = Position + Direction	**Invalid**
(viii) Direction = Scalar * Direction	**Valid**

f. **(11 points)** We'll use the following notation for 2D transformation matrices:

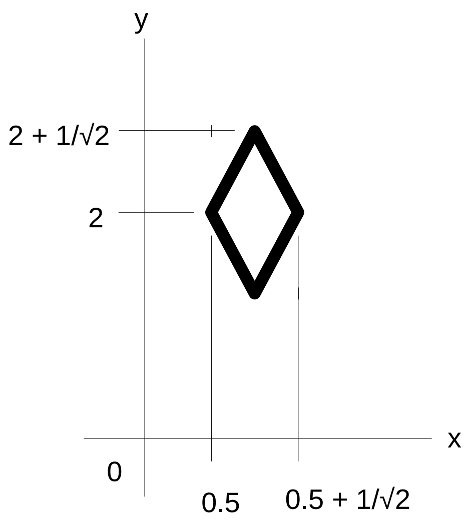
T(*x*, *y*): Translates a point by (*x*, *y*)
R(*a*): Rotates anti-clockwise around the origin by *a* degrees
S(*sx*, *sy*): Scales the coordinates of a point, relative to the origin, by (*sx*, *sy*)

You are given a unit square with minimum corner (0, 0), as shown below:



(i) **(3 points)** Sketch the final result of applying M = S(0.5, 1) × T(1, 2) × R(–45) to the square. Remember: we apply transforms right-to-left. Don't forget to draw the coordinate axes, with necessary markings

Solution:



(ii) **(3 points)** Using the notation above (T, R and S), and <u>without</u> explicitly using an inverse operation (e.g. $T^{-1}(x, y)$), write an expression for the inverse of the transformation M.

Solution: R(45) × T(–1, –2) × S(2, 1)

(iii) **(3 points)** Write the three 3x3 matrices representing S(0.5, 1), T(1, 2) and R(–45) in homogeneous coordinates.

Solution:

$$S(0.5,1) = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T(1,2) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R(-45) = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(iv) **(2 points)** Bianca wants to find the distance between the transformed square and the point (10, 10). To do this, she multiplies (10, 10) by the inverse transformation obtained in (ii) to transform it to the local frame of the square, and then computes the distance in this frame. Is Bianca correct in doing this? Why or why not?

Solution: No, this is incorrect. The non-unit scaling of 0.5 along X messes up the distances. If all transforms in the chain were rigid (rotations or translations), things would be ok.

4. **Real-Time Graphics and the GPU (12 points):**

   a. (3 points) The *painter's algorithm* for hidden surface removal is an alternative to the Z-buffer algorithm that instead sorts scene objects by distance and then draws them starting from the furthest away, with each overwriting (or "overpainting") what has previously been drawn. Describe two situations where this algorithm will not work well.

   Solution (other answers possible):

   (1) Lots of polygons overlapping the same point when projected (too much overdrawing)

   (2) Two polygons overlapping in depth (need to be split for a well-defined sorted order)

   b. (2 points) What will happen if you use the following OpenGL code for drawing?

```
void drawSquare()
{
  glVertex3f( 0,  0, 0); glVertex3f(10,  0, 0);
  glVertex3f(10, 10, 0); glVertex3f( 0 ,10, 0);
}

void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW); glLoadIdentity();

  glBegin(GL_QUADS);
  glColor3f(0.0, 1.0, 0.0); drawSquare();
  glTranslatef(10, 0, 0);
  glColor3f(1.0, 0.0, 0.0); drawSquare();
  glEnd();
}
```

   Solution: Calling glTranslatef, which changes the modelview matrix, is invalid within a glBegin/glEnd block. The code will (typically) silently throw a GL_INVALID_OPERATION error and ignore the command. (*It does NOT produce two side-by-side squares.)*

   c. (1 point) Modern GPUs have unified shader cores, instead of separate cores for fragment processing and vertex processing. Why is this a good thing? Clearly describe one advantage.

   Solution: Vertices and fragments are processed one after the other (first vertices, then fragments), so the same circuitry can be reused for vertex and fragment processing, increasing the amount of die area doing active computation at any given time.

d. **(6 points)** A hierarchical Z-buffer extends the standard Z-buffer by adding a pyramid of downsampled versions of it. According to its inventors:

"The basic idea of the Z pyramid is to use the original Z buffer as the finest level in the pyramid and then combine four Z values at each level into one Z value at the next coarser level by choosing the farthest Z from the observer. Every entry in the pyramid therefore represents the farthest Z for a square area of the Z buffer. At the coarsest level of the pyramid there is a single Z value which is the farthest Z from the observer in the whole image."

The hierarchical Z-buffer allows a large occluded object to be rejected quickly, without generating and rejecting each of its fragments individually. Describe in detail (pseudocode is ok) how this test would work. Assume you are testing a single large polygon and you want to check, as quickly as possible, if it is completely occluded or not. You don't need to provide geometric calculations, just indicate the general flow of logic.

Solution:

```
Algorithm isOccluded(Z_buffer_node N, Polygon P)

        Let B be the bounding box of N on the screen
        Let Z be the smallest depth value of P within B (infinity if P outside B)

        if Z >= N.maxZ
                return true

        if N has no children
                return false

        For i = 0 to 3
                if (!isOccluded(N.child[i], P))
                        return false

        return true
```

(Several folks lost points for omitting "within B". It doesn't work if Z is the smallest depth of the whole polygon, or of just its vertices.)

5. **Raytracing and Largescale Rendering (6 points):**

   a. **(2 points)** Chinmay is writing a renderer which will draw scenes using a perspective projection. To accelerate rendering of very large scenes, he plans to use a level of detail hierarchy. Arrange these representations in order of increasing distance (from the observer) at which they should be used:

   single impostor

   environment map

   simplified mesh with 50K polygons

   collection of impostors

   original mesh with 1M polygons

   simplified mesh with 500 polygons

   Solution:

   original mesh with 1M polygons

   simplified mesh with 50K polygons

   simplified mesh with 500 polygons

   collection of impostors

   single impostor

   environment map

   b. **(2 points)** Motion blur occurs when the object being photographed moves across the frame while the shutter is open. Explain how this effect can be simulated in a raytracer.

   Solution: Distribute rays over time. *1 point for saying blend entire frames rendered at different times (ideally, there should be some independent jittering in the time domain of rays at different pixels).*

   c. **(2 points)** State one visual effect that standard backwards raytracing is ideally suited to handle, and one effect that it is very poor at handling. No need for explanations.

   Solution (several valid answers):

   Good: Specular (mirror) reflection
   Bad: Diffuse inter-reflections

6. **Polygon Meshes (4 points):** Recall that a polygon mesh consists of a set of polygonal faces, joined to each other at edges. Two faces are said to be *connected* to each other if there is an unbroken sequence of faces between them, where each successive pair of faces is joined along an edge. A *connected component* of a mesh is a set of faces in which every pair is connected. A *maximal* connected component is one that cannot be enlarged by adding a new face. Describe a reasonably efficient algorithm to find all maximal connected components of a mesh.

   Solution:

   a. Put all faces in a set S (e.g. by having an actual set, or by adding a boolean flag to each face) and start a new connected component.
   b. Do depth first search (DFS) on the graph whose nodes are mesh faces, and edges are every pair of adjacent faces. Start from an arbitrary face drawn from S. For every face visited by DFS, remove it from S and add it to the current connected component.
   c. If S has any faces left, start a new connected component and repeat Step (b).

7. **Antialiasing (2 points):** Succinctly describe the difference between pre-filtering and post-filtering to reduce aliasing when rendering an object with a fine textured pattern.

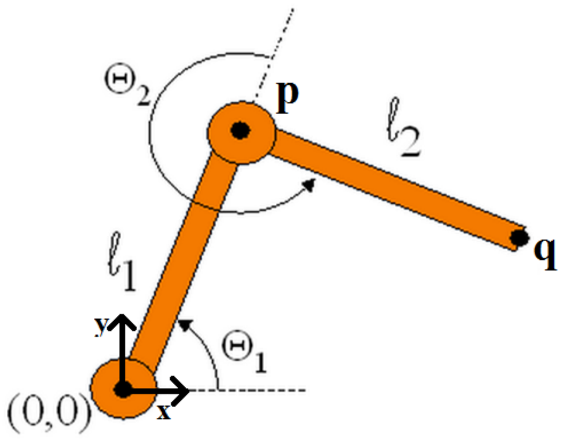
   Solution:

   Pre-filtering: Downsample the texture using a lowpass filter to roughly match its resolution after projection onto the screen, BEFORE rendering.

   Post-filtering: Render the scene at an extra-high resolution and downsample it using a lowpass filter to output resolution, AFTER rendering.

   (Too many people answered the question in general terms, not for the specific scenario mentioned in the question. And you do need to mention downsampling at the end of the post-filtering step, that is the actual filtering part.)

8. **Forward Kinematics (5 points):** Suppose that you have the following robotic arm (see figure below). Its base is affixed to the origin of the world coordinate system. The first segment has length l_1 and can rotate freely about the origin by an angle θ_1 wrt to the x-axis. The second segment has length l_2 and can rotate freely by an angle θ_2 about the joint ***p*** that connects the first and second segments. Compute the 2D position of the end joint ***q*** in the world coordinate system, in terms of l_1, l_2, θ_1, θ_2. Show your derivation (you get no points without the derivation).



Solution:

From coordinate geometry, we first derive the position of **p**:

$$p_x = l_1 \times \cos(\theta_1)$$
$$p_y = l_1 \times \sin(\theta_1)$$

From this, we can obtain the position of **q** by adding the offset of the second link:

$$q_x = p_x + l_2 \times \cos(\theta_1 + \theta_2)$$
$$q_y = p_y + l_2 \times \sin(\theta_1 + \theta_2)$$

Substituting, we get the final result:

$$q_x = l_1 \times \cos(\theta_1) + l_2 \times \cos(\theta_1 + \theta_2)$$
$$q_y = l_1 \times \sin(\theta_1) + l_2 \times \sin(\theta_1 + \theta_2)$$

9. **Inverse Kinematics (6 points):**

   a. (3 points) You are trying to position the end effector of a jointed robot at a target location in 2D, as we studied in class. Assuming a solution exists, is Cyclic Coordinate Descent guaranteed to converge to some solution? Prove (informally) your answer.

   Solution: Yes. The distance to the target never decreases in any iteration, so (by the result that an increasing sequence of real numbers which is bounded above converges to its supremum), the process is convergent.

   Caveat: Technically, it's possible the process gets stuck in a local minimum in some degenerate situations. In this case, the arm doesn't reach the target and hence this is not a solution (even if a valid solution exists), although the process is still convergent. If you made this observation with a valid degenerate case, you should get full credit.

   b. (3 points) You are animating a character by setting the positions and orientations of <u>both</u> its hands, in 3D. The character has 25 joints. You plan to use the Jacobian method for inverse kinematics. How many rows, and how many columns, does the Jacobian have? Briefly explain how you arrived at your answers.

   Solution:

   (This question was (unintentionally) ill-posed. The number of degrees of freedom (angle parameters) at each joint was not specified. Answers that assumed the total number of joint angles was 25, as well as those that noted the lack of information, got full credit.)

   The constraint vector X has 12 parameters (3 rotation and 3 translation parameters for each hand). if we assume the total number of joint angles is 25 (see note above), the Jacobian has 12 rows and 25 columns.

10. **Extra (1 point):** Suggest an interesting new assignment for this course.

    Thanks for your feedback and the interesting suggestions!