CSC418H1F Computer Graphics, Fall 2010

Assignment 2 (15% of course grade)

Part A (written): Due before tutorial (6:09pm) on Wednesday, October 27th.
Part B (programming): Due 11:59pm on Friday, November 5th.

Part A [50 marks in total]

As in Assignment 1, the questions below require thought, so you are advised to consult the relevant sections of the textbook, the online lecture notes and slides, and your notes from class well in advance of the due date. Your proofs and derivations should be clearly written, mathematically correct, and concise.

*All questions require showing the steps toward the solution, and marks will be subtracted if this is not the case. Even if you cannot answer a question completely, it is very important that you show your (partial) answers and your reasoning. Otherwise your TA will **not** be able to award you partial marks.*

You can either hand in your handwritten solutions to Part A to your TA before the tutorial, or you can submit them online in PDF format by the due date/time (e.g., using a scanner to scan your hand-written answers or a typesetting program such as Word or LaTeX).

1. **Parallelism & Perspective Projection** [18 marks]

   Perspective projection preserves linearity but does not preserve parallelism. Indeed, the perspective projection of two parallel 3D lines will intersect at a 2D point called the *vanishing point*, and this point will generally *not* be at infinity.

   (a) [9 marks] Let

   $$l_1(\lambda) = \bar{p}_1 + \vec{d}\lambda, \quad \lambda \in \mathbb{R}, \text{ and}$$
   $$l_2(\lambda) = \bar{p}_2 + \vec{d}\lambda, \quad \lambda \in \mathbb{R}$$

   be the parametric equations of two 3D lines, where $\bar{p}_1, \bar{p}_2$ are 3D points and $\vec{d}$ is a 3D vector, all represented in the camera's 3D reference frame. Assuming that the camera's focal length is f and its image plane is aligned with the xy-plane, give an expression for the homogeneous coordinates of the lines' vanishing point.

   (b) [4 marks] Even though it does not happen in general, there are special cases of parallel 3D lines that do project to parallel lines under perspective projection. Give an orientation vector $\vec{d}$ for which the lines in part (a) project to parallel lines.

   (c) [5 marks] Let

   $$\pi_1(\lambda, \mu) = \bar{p}_1 + \vec{d}\lambda + \vec{d'}\mu, \quad \lambda, \mu \in \mathbb{R}, \text{ and}$$
   $$\pi_2(\lambda, \mu) = \bar{p}_2 + \vec{d}\lambda + \vec{d'}\mu, \quad \lambda, \mu \in \mathbb{R}$$

   be the parametric equations of two parallel 3D planes, with $\bar{p}_1, \bar{p}_2$ and $\vec{d}$ defined as in (a) and $\vec{d'}$ another 3D vector. Give an expression for the projection of the line of intersection of these two planes. This line is usually called the *vanishing line* and, when the planes are parallel to the ground, it coincides with the horizon. *Hint: Use your solution for part (a).*

2. **Surfaces of Revolution, Normals & Tangents** [10 marks]

   Many real-life objects can be modeled as surfaces of revolution. For example, a variety of *vase* forms can be represented by rotating about the z axis the following parametric curve:

$$\bar{p}(t) \;=\; \left(a + t^2 cos(t),\ 0,\ \frac{t}{b}\right), \quad 0 \le t < 2\pi \;, \qquad (1)$$

   where a and b are real-valued constants.

   (a) [3 marks] Give the parametric equation for the resulting surface of revolution.

   (b) [4 marks] Give the equation of the tangent plane of the surface at a point, as a function of the surface parameters.

   (c) [3 marks] Give an expression for the unit normal of the surface at a point, as a function of the surface parameters.

3. **Cameras & 3D Viewing** [10 marks]

   Consider a scene that has two cameras, *ONE* and *TWO*. The origin of camera *ONE* is at point $\bar{e}_1$ in the world coordinate frame, and its x, y and z axes are aligned with vectors $\vec{u}_1, \vec{v}_1$ and $\vec{w}_1$, respectively. Camera *TWO*'s coordinate frame, however, is defined with respect to camera *ONE*: its origin is $\bar{e}_2$ and its coordinate axes are along vectors $\vec{u}_2, \vec{v}_2, \vec{w}_2$, all expressed in camera *ONE*'s coordinate frame.

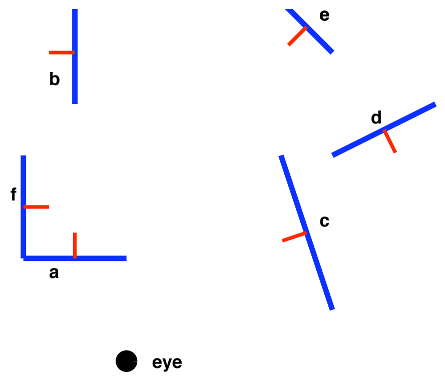
   (a) [3 marks] What is the origin of camera *TWO* in world coordinates?

   (b) [3 marks] Give the transformation that maps world coordinates to coordinates expressed in camera *TWO*'s reference frame.

   (c) [4 marks] Assuming that camera *TWO* has focal length f and that its image plane is the xy-plane of its coordinate frame, give the projection matrix that maps 3D points in world coordinates to 2D points in camera-*TWO*-centered coordinates.

4. **Visibility** [12 marks]

   The illustration below shows a top-down view of a 2D scene, with segments in blue and outward normals in red; the eye location is illustrated by the black dot.



(a) [3 marks] Assuming the particular scene and camera placement shown above, is it possible to exclude any segments from rendering? If so explain why; if not, explain why not.

(b) [5 marks] Draw a BSP tree for the above scene by adding the segments in the labeled order (*i.e.* from 'a' to 'f').

(c) [4 marks] Indicate how your tree will be traversed when rendering the scene from the eye location specified.

Turning in your solutions to Part A online

To submit solution to Part A online, first make sure that the contents of the PDF file are legible, then use the following command:

```
submit -c A2a csc418h -a A2a A2a_solution.pdf
```

[Continues on next page ⟶]

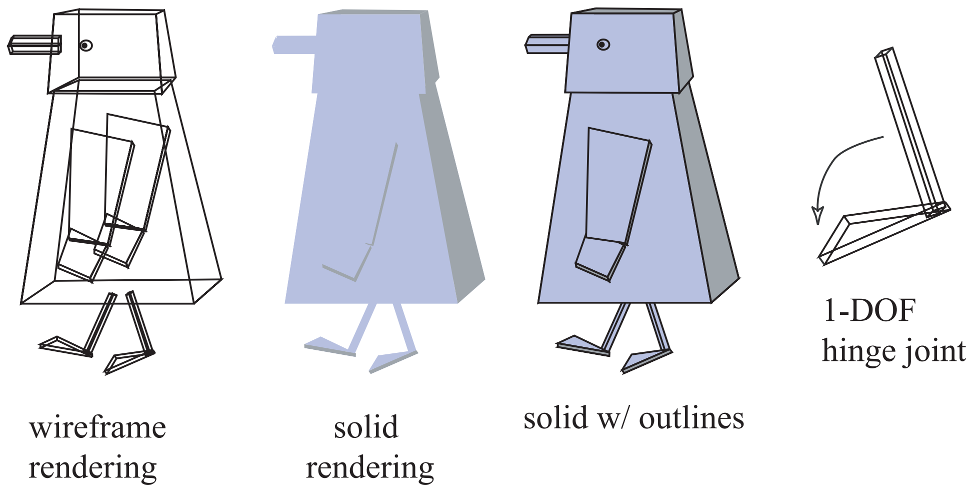


Figure 1: 3D Robot penguin.

Part B: Animating & Rendering a 3D Object [50 marks in total]

In this assignment, you will create a simple keyframe animation tool, and use it to animate a 3D character.

3D geometry and DOFs

You will first define a penguin in 3D as in Figure 1. The penguin will have 24 *degrees-of-freedom* (DOFs) that define the pose of the penguin at any time. In assignment 1, the DOFs were just the 2D rotation angles of the penguin's joints and the opening widths of the beak.

For this assignment, the DOFs will be as follows. The penguin's knee and elbow joints are 1 DOF *hinge* joints that can only rotate forward and backward; the penguin's beak has a single open-close DOF (as before); the hip and shoulder joints each have 3 angles of rotation; the penguin's head has one degree of freedom so that it can rotate around the neck axis. Finally, the penguin can globally translate and rotate in space, i.e., a translation/rotation is applied to the entire penguin. (This adds 3 DOFs for translation and 3 more for rotation.) Rotation should be about the penguin's (approximate) centroid, regardless of the translation.

(a) [10 marks] Design the parts in terms of suitable shapes and deformations, and draw them using OpenGL. Your penguin does not need to look exactly like the one in the figure (e.g., the geometry can be more complicated). You do not need to draw circles for joints; drawing eyes or other decorations is optional, but they should not obscure the shape and motion of the character.

Rendering style

Your program should include five rendering modes: wireframe, solid, solid with outlines, "shiny metal," and "matte:"

(b) [10 marks] *Basic polygon rendering styles:* The first three rendering styles are shown in the figure. Solid and wireframe can be created using the same geometry routines (e.g., `glVertex` calls), but with different settings for `glPolygonMode`. Solid can be drawn together with outlines, by rendering wireframes fill after the solid fill has been rendered within the display function, but using `glPolygonOffset` on the wireframe rendering. (See the OpenGL reference books for information on these routines). The user should be able to choose the rendering style in the user interface. Note that in general, it is not required that the outlines are perfectly crisp; drawing them perfectly is not easy in OpenGL, or required for this assignment.

(c) [5 marks] *Material properties:* In addition to the above styles, the user should also be able to draw the penguin using solid polygons with a "metallic" or "matte" appearance. To do this, you will need to assign material properties to your polygons via `glMaterial` and define a light source using `glLight`. Again, refer to the manual for information on how to set up light sources and shading in OpenGL.

(d) [10 marks] *Light source control:* You should add controls to the user interface to allow the user to control the position of the light source around the object. You should define the light source position to be sufficiently far away from the object so that it does not interfere with the object's motion. For the purposes of the assignment it suffices to move the light source on a circle parallel to the xy-plane, of a sufficiently large radius.

Keyframe Animation

The second goal in this part of the assignment is to create a keyframe animation tool that allows you to set *keyframes*, and then play back an animation that interpolates those keyframes. The values of the DOFs at each frame will be defined by a function $\bar{q}(t)$ that defines the values of the DOFs at each time t. This function will be defined by interpolating keyframes. Each keyframe comprises a time and a pose, that is, a pair $(t_i, \bar{q}_i)$, where the pose $\bar{q}_i$ is a 24-dimensional vector containing the DOFs at time t_i. That is, $\bar{q}_i$ contains the value of each joint angle, the gripper distances, and the global translation and rotation of the torso at time t_i.

(e) [5 marks] The animation tool should work as follows. Each DOF in the character will correspond to a "spinner" in the user interface; adjusting a slider will change the corresponding DOF for the character shown in the display window. Pressing the button "Update Keyframe" sets the current pose as a keyframe in the animation system, and then pressing "Start/Stop animation" plays back the animation. Use of the starter code is required. Be sure to include a sample set of keyframes with your electronic submission.

(f) [5 marks] When you play back the animation, the smoothly varying pose $\bar{q}(t)$ should be computed by Catmull-Rom interpolation of the keyframes (this interpolation will be covered in the next tutorial).

Helper Code

We have provided helper code that has many of the data structures and user interface elements already in place. The compiled skeleton code will display a solid shaded cube located at the origin. You can move the cube using the UI (Root→translate x/y/z). A sample keyframe file has been provided and can be loaded to demonstrate the animation and render-to-file capabilities. Your main job is to define the procedures for rendering the character at a given time, and for computing the interpolation. See the code for details.

To unpack, compile and run this demo on CDF, download the file *a2.tgz* and use the following commands

```
tar xvfz a2.tgz
cd a2/penguin3d
make
penguin
```

Tips

There are several different parts to this assignment, and it will be easiest to tackle them one at a time — and debug each part in turn — rather than trying to implement everything at once. First, design the animal and the joint angles. We recommend that you render each polygon with a solid color, perhaps a different color for each face. Make sure you've got Z-buffering working properly so that visibility is correct. Make sure that the character moves properly when adjusting the controls. Be careful when designing rotations. If you represent an angle between 0 and 360 degrees, then do not animate angles near 0 since that might lead to a discontinuity. Similarly, if you represent angles between -180 and 180 degrees then you will be best to keep your angles, on average, near zero rather than near 180 or -180 degrees. We suggest you enable lighting and tackle the last two rendering styles (metallic, matte) only after you are done with everything else.

Turning in your Solution to Part B

All your code should remain in the directory a2/penguin3d. In addition to your code, you must complete the files *CHECKLIST* and *REPORT* contained in that directory. *Failure to complete these files will result in zero marks on your assignment.*

(g) [5 marks] The *REPORT* file should be a well-structured written (or diagramatic) explanation of the basic components of your implementation. The description should be a clear and concise guide to the concepts, not a simple documentation of the code.

Note that this file should *not* be thought of as a substitute for putting detailed comments in your code. Your code should be well commented if you want to receive full (or even partial) credit for it.

To pack and submit your solution, execute the following commands from the directory containing your code (i.e., *a2/penguin3d*):

```
cd ../../
tar cvfz a2-solution.tgz a2
submit -c csc418h -a A2b a2-solution.tgz   (if registered for CSC418)
submit -c csc2504h -a A2b a2-solution.tgz  (if registered for CSC2504)
```