Homework 1

Due: Wed, March 28, 11:00pm. Submit through ELMS as a pdf file. It can either be distilled from a typeset document or handwritten, scanned, and enhanced (using an app like CamSanner). Beware that scanned homeworks that are not clearly legible may be returned to you, requiring rescanning and possibly recopying. Avoid writing on both sides of the paper, since the writing can bleed through to the other side.

Late policy: Up to 6 hours late: 5% of the total; up to 24 hours late: 10%, and then 20% for each additional 24 hours.

Problem 1. Short answer questions.

(a) Give two examples that might arise in a game implemented in Unity, one where you want a rigid-body to have a *collider* and one where you want a *trigger*.

(b) You have a triangle in the plane defined by three points a, b, and c (see Fig. 1). You would like to interpolate 5 points $p[0], \ldots, p[4]$ where $p[0] = a$ and $p[4]$ is the midpoint between b and c. For $0 \leq i \leq 4$, give a formula that expresses $p[i]$ as an affine combination of the points a, b, and c. (You may introduce additional temporary points if you like, but express them as affine combinations of the original points.)

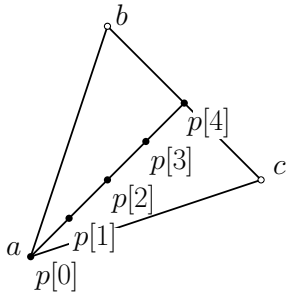
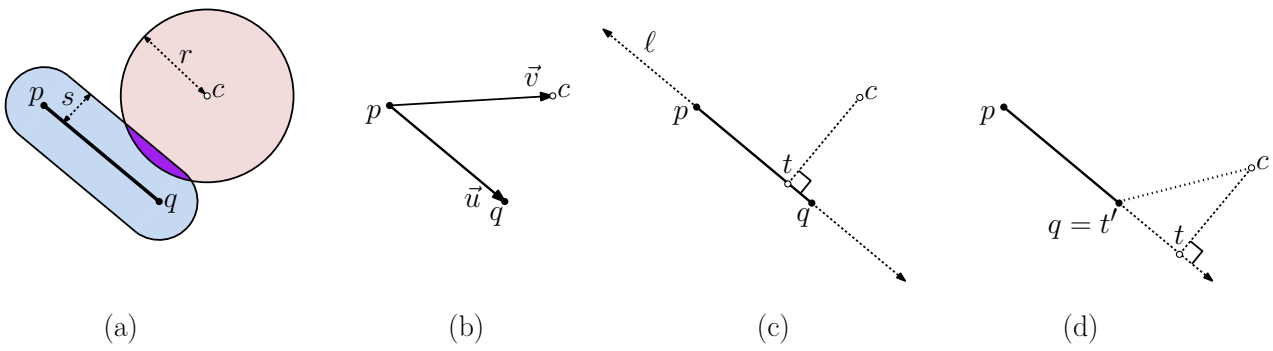


Figure 1: Problem 1(b): Affine combinations.

(c) In Unity, you want to rotate an object through 90° degrees to occur smoothly over a period of 4 seconds. In your Update function, how many degrees of rotation should you apply? (Hint: Use the value of Time.deltaTime.)

Problem 2. Your first assignment in your new job working for a game-engine company is to design a function that determines whether a sphere collider and a capsule collider intersect in 3-dimensional space. We will walk through the steps to obtain the mathematical conditions behind this test and then derive a C# function to do it.

The sphere collider has center $c = (c_x, c_y, c_z)$ and radius r (see Fig. 2(a)). It consists of all the points whose distance from c is at most r. The capsule collider is given by three quantities: $p = (p_x, p_y, p_z)$ and $q = (q_x, q_y, q_z)$ are the endpoints of the central axis of the collider and s is its radius. The collider consists of all the points whose distance from the line segment $\overline{pq}$ is at most s. Let us make the simplifying assumptions that p, q, and c are all distinct points, and that c does not line on the line passing through p and q.



(a) (b) (c) (d)

Figure 2: Problem 2: Sphere/Capsule collision.

Define $\vec{u}$ to be the vector directed from p to q (see Fig. 2(b)), and define $\vec{v}$ to be the vector directed from p to c. Note that these should *not* be normalized to unit length. For example, $\|\vec{u}\|$ should equal the length of the segment $\overline{pq}$.

(a) Using the operations of affine and Euclidean geometry, explain how to compute $\vec{u}$ and $\vec{v}$ from p, q, and c. (Hint: If this seems easy, it is. We're just getting started.)

Consider the infinite line ℓ passing through p and q. Let t be the point on ℓ such that the line segment $\overline{tc}$ is perpendicular to ℓ (see Fig. 2(c) and (d)). (Note that t may generally lie outside the line segment $\overline{pq}$, and this is fine for now.)

(b) Using the operations affine geometry and Euclidean geometry, explain how to compute the point t. (Hint: The vectors $\vec{u}$ and $\vec{v}$ from part (a) should come in handy.)

It is a fact (which you do *not* need to prove) that if t lies within the line segment $\overline{pq}$ then it is the closest point on this segment to c. If not, the closest point to c is one of the endpoints, p or q.

(c) Explain how to modify your solution to (b) to determine the point t' on the line segment $\overline{pq}$ that is closest to c. (Hint: Just explain how to modify your solution to (b) to achieve this.)

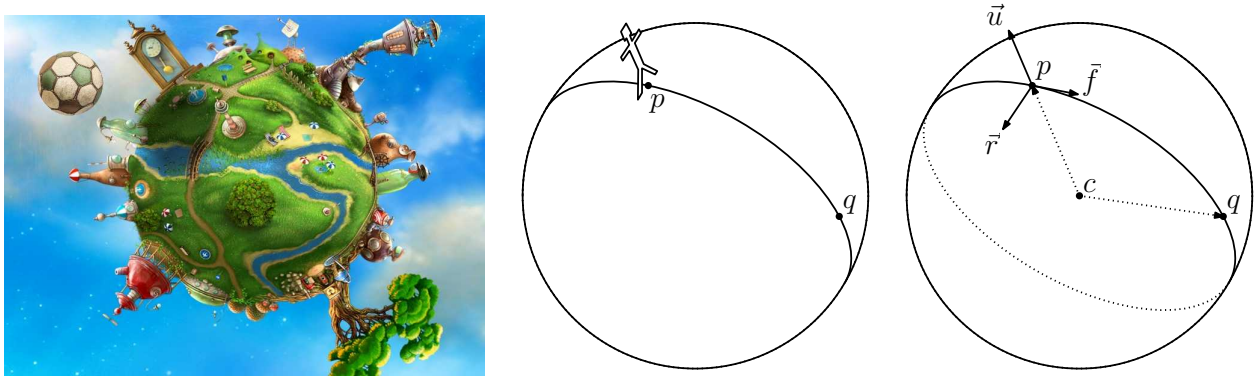
(d) Combining the results of the previous parts, present a test (in mathematical notation) that determines whether the two colliders intersect. (Note: The boundaries do not need to intersect. If one collider is completely contained within the other, they are still considered to intersect.)

(e) Present a C# function with the following signature that implements your intersection test:

```
bool SphereCapsuleCollide(Vector3 c, float r, Vector3 p, Vector3 q, float s)
```

You may make use of the basic math/geometry functions that Unity provides (e.g., Vector3 methods or Mathf functions), but you *cannot* use more advanced functions (e.g., Unity's CapsuleCollider class methods). Exact syntactic correctness is not essential, but your program should be close to valid C# code.

Problem 3. Inspired by Tiny Planet videos and similar games (see Fig. 3(a)), you have decided to create a game where a player moves around the surface of a sphere. In order to implement your idea, you will need to define a coordinate frame for the player object.



(a) (b) (c)

Figure 3: Problem 3: Tiny-planet coordinate frame.

Let us assume that the sphere is centered at a point c, and it has a radius of s units. The player's frame will be centered at some point p that lies on the sphere. In order to indicate the direction in which the player is facing, a second point $q \neq p$ is given on the surface of the sphere (see Fig. 3(b)). These two points define a great circle on the sphere. The player is situated so that its vertical axis is aligned with the vector from c to p, and its forward-looking vector is directed tangent to the sphere at point p along the smaller arc of the great circle containing p and q (see Fig. 3(c)). (Let us make the simplifying assumption that p and q are not polar opposites from each other, for otherwise there are infinitely many great circles through p and q.)

The objective of this problem is to define an orthonormal coordinate frame for the player object. The origin of this frame is p. The basis vectors are:

- $\vec{u}$: "up" relative to the player (along the ray from c through p)
- $\vec{f}$: "forward" relative to the player (tangent to the great circle through p and q)
- $\vec{r}$: "right" relative to the player (perpendicular to both $\vec{u}$ and $\vec{f}$)

(a) Explain how to compute the above three basis vectors using the operations of affine and Euclidean geometry. Be sure that the final vectors are normalized to unit length.

Hint: Trigonometry and calculus are *not* needed to solve this problem. This can all be done using the standard basic of Euclidean geometry (such as affine combinations, dot product, and cross product). It is easiest to compute these in the order $\vec{u}$, then $\vec{r}$, then $\vec{f}$.

(b) Present a C# function with the following signature that implements your construction:

```
void PlayerFrame(Vector3 c, float s, Vector3 p, Vector3 q,
                 out Vector3 u, out Vector3 f, out Vector3 r)
```

You may make use of the whatever standard math/geometry functions that Unity provides.

Note: In your answers to parts (a) and (b), state *explicitly* whether you are assuming a *right-handed* or *left-handed* world coordinate system. Standard math textbooks assume a right-handed system, meaning that positive rotations are counterclockwise and the cross-product follows the right-hand rule. However, Unity uses a left-handed system, and this implies that positive rotations are clockwise and the cross-product follows the left-hand rule. For example, see https://docs.unity3d.com/ScriptReference/Vector3.Cross.html.

Problem 4. Consider the series of shapes $S_0, S_1, \ldots$ shown in Fig. 4 below. With each step, each line segment of length x is replaced by four segments each of length $x\sqrt{2}/4$. Let $S^* = \lim_{i\to\infty} S_i$.

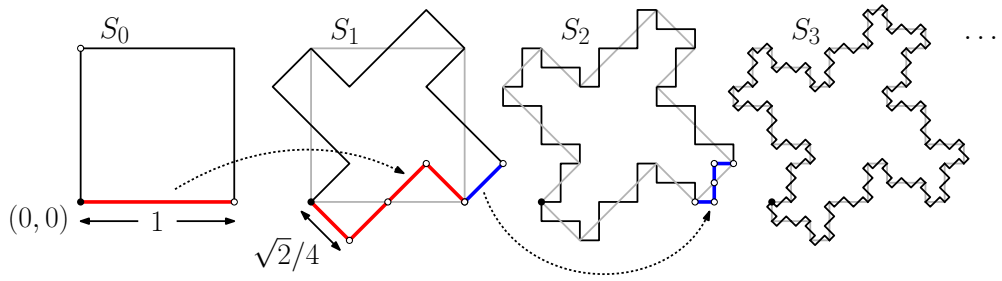


Figure 4: Problem 4: Fractals and L-systems.

(a) Derive the fractal dimension of the *boundary* of S^*. You may express your answer as the ratio of two logarithms.

(b) Derive an L-system to generate this shape. (For the sake of consistency, we recommend that you start in the lower-left corner of the unit square and proceed counterclockwise around the boundary.) In particular, answer the following:

(i) What are the values of the step-size d and angle increment δ? These values may change from one shape to the next. If so, let d_i and δ_i denote the step size and angle increment for generating S_i. Assume that the side length of S_0 is 1.

(ii) What are the variables (symbols) V of your L-system?

(iii) What is the start symbol (or start string), ω, for your system?

(iv) What are the production rules, P? (Hint: Be sure that at the end of each sequence, your current state is properly oriented to start the next sequence.)

(For guidance on solving this, see March 10 update of Lecture 11 on fractals and L-systems.

Problem 5. For your new game, you will be animating a model of a tower crane in 2-dimensional space. The crane's parts are illustrated in Fig. 5(a). The structure sits on a base, which we take to be the origin of the crane's coordinate system.

The vertical *mast* rises upwards H units to a point where the operator's booth sits. From here there is a horizontal *jib* upon which slides a *trolley* at a distance of W units from the operator. The *hook* is suspended D units beneath the trolley. (Increasing the value of D causes the

hook to drop lower.) The reference (or bind) pose is such that $H_0 = 130$, $W_0 = 100$, and $D_0 = 70$. There are four coordinate frames as shown in Fig. 5(b): (a) the base, (b) the operator, (c) the trolley, and (d) the hook.

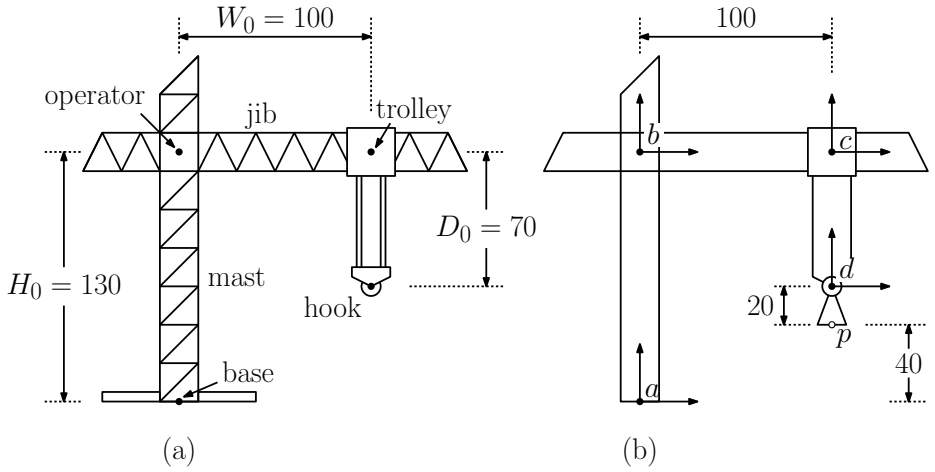


Figure 5: Problem 5: Animating a tower crane.

(a) Following the naming convention for the local pose transformations (given in Lecture 9) express the following local pose transformations a 3×3 homogeneous matrices.

(i) $T_{[c \leftarrow d]}$, which translates coordinates in the hook frame to the trolley frame.

(ii) $T_{[b \leftarrow c]}$, which translates coordinates in the trolley frame to the operator frame.

(iii) $T_{[a \leftarrow b]}$, which translates coordinates in the operator frame to the base frame.

(Hint: It may help to recall the form of a translation matrix from Lecture 6.)

For example, a point p located 20 units below the hook would be represented in the hook frame as $p_{[d]} = (0, -20, 1)$. Its coordinates with respect to the trolley would be $p_{[c]} = (0, -90, 1)$. Thus, the transformation $T_{[c \leftarrow d]}$ should satisfy

$$T_{[c \leftarrow d]} \cdot \begin{pmatrix} 0 \\ -20 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -90 \\ 1 \end{pmatrix}.$$

(b) Show that by multiplying these matrices, we obtain a matrix $T_{[a \leftarrow d]}$ that maps point p into its coordinates with respect to the base frame. Verify your resulting matrix by showing that

$$T_{[a \leftarrow d]} \cdot p_{[d]} = p_{[a]} = \begin{pmatrix} 100 \\ 40 \\ 1 \end{pmatrix}.$$