Solutions to the First Practice Midterm Problems

Disclaimer: These solutions have not been fully checked for correctness. If anything seems fishy, check with me (mount@umd.edu).

Solution 1:

(a) (i) and (iii) are true. Declaring an object to be *static* informs Unity that the object's position is fixed. This means that navigation, physics, and lighting can be optimized based on the fact that the object's location does not change. (ii) is false: Even though the object does not move, it can still interact dynamically with its environment. (iv) is also false: You can instantiate many instances of a static game object. (The term "static" refers to its position. It is *not* related to static variables in programming languages.)

(b) A right-handed system is characterized by the relation $\vec{u}_x \times \vec{u}_y = \vec{u}_z$. (In a left-handed system, the result of the cross product is $-\vec{u}_z$.)

(c) (i) and (iii) may provide poor approximations. (i) AABBs will yield a poor approximation if the thin object is oriented diagonally, (iii) A sphere is never a good approximation to a thin object.

(d) Task I is an example of *forward kinematics*, and Task II is an example of *inverse kinematics*. Inverse kinematics is more challenging for a number of reasons. First, for a given set of constraints, a solution might not even exist. Also, there can be be multiple (in fact infinitely many) solutions. Second, while forward kinematics involves simple matrix multiplications, inverse kinematics generally involves solving a nonlinear numerical optimization problem.

(e) Declare the object to be *kinematic* (e.g., by selecting the IsKinematic option in the Inspector window).

(f) Unlike Euler angles, quaternions are intrinsic, that is, they do not depend on the choice of coordinate system, and this means that interpolations are more natural.

Solution 2:

(a) In order to determine the equation of the collider plane, we observe that a point $q = (x, y, z)$ lies on the plane if and only if the free vector from p to q, that is, $q - p$, is orthogonal to $\vec{u}$. Letting $p = (p_x, p_y, p_z)$ and $\vec{u} = (u_x, u_y, u_z)$, the vector $q - p$ has the coordinates $(x - p_x, y - p_y, z - p_z)$. This is orthogonal to $\vec{u}$ if and only if the dot product of these two vectors is zero, that is, $\vec{u} \cdot (q - p) = 0$. We can express this explicitly in terms of the coordinates as

$$\begin{aligned} 0 &= \vec{u} \cdot (q - p) = (u_x, u_y, u_z) \cdot (x - p_x, y - p_y, z - p_z) \\ &= u_x(x - p_x) + u_y(y - p_y) + u_z(z - p_z) \\ &= u_x x + u_y y + u_z z - (u_x p_x + u_y p_y + u_z p_z). \end{aligned}$$

It follows that the equation of the collider plane is $\alpha x + \beta y + \gamma z + \delta = 0$, where $\alpha = u_x$, $\beta = u_y$, $\gamma = u_z$, and $\delta = -(u_x p_x + u_y p_y + u_z p_z)$.

(b) Any point on the infinite line $\overleftrightarrow{ab}$ can be expressed as $q(t) = (1-t)a + tb$, for some real t. To determine the intersection with the collider plane, we plug $q(t)$ into the plane equation from (a) and solve for t. We have

$$0 \;=\; \vec{u} \cdot (q(t) - p) \;=\; \vec{u} \cdot ((1-t)a + tb - p) \;=\; \vec{u} \cdot (t(b-a) - (p-a)).$$

By the linearity of the dot product operator, we can express this as

$$0 \;=\; t(\vec{u} \cdot (b-a)) - \vec{u} \cdot (p-a) \qquad \text{or equivalently} \qquad t \;=\; \frac{\vec{u} \cdot (p-a)}{\vec{u} \cdot (b-a)}.$$

To obtain the equivalent expression in terms of the coordinates, we can expand this:

$$t \;=\; \frac{u_x(p_x - a_x) + u_y(p_y - a_y) + u_z(p_z - a_z)}{u_x(b_x - a_x) + u_y(b_y - a_y) + u_z(b_z - a_z)}.$$

If the divisor is nonzero (see part (c)), then the value of t is well defined. If $0 \le t \le 1$, the affine combination is a convex combination, and $q(t)$ lies on the line segment $\overline{ab}$. If not, both points either lie above the plane or both lie below it, and we can declare that the segment does not intersect the collider.

(c) Before computing t in part (b), we should first check whether the divisor is zero. There are two reasons why this might happen:

(1) It might be that either $\vec{u}$ or $b - a$ is the zero vector. However, this cannot happen. The problem description stipulates that $\vec{u}$ is a vector of unit length, and $b \neq a$. Therefore, neither is the zero vector.

(2) Both vectors are nonzero, but $\vec{u}$ is orthogonal to $b - a$. Equivalently, the line $\overleftrightarrow{ab}$ is parallel to the collider plane, and hence it either lies in the plane or never intersects it. This possibility is not explicitly ruled out by the problem description.

Although the problem does not ask you to fix it, here is a hint. If the points lie on opposite sides of the collider plane, then these two vectors cannot be orthogonal. Thus, there are three possibilities—both points lie above the collider plane, both lie below it, or both lie on it. The last case is disallowed by the problem description, but the other two could happen. You can determine this by the formula given in part (a). Observe that if it does happen, the line segment cannot hit the collider.

(d) Assuming that part (b) succeeds in the sense that t is well defined and $0 \le t \le 1$, we know that $q(t)$ lies on the collider plane. All that remains is testing that the distance between $q(t)$ and p is at most r, the disk's radius. Recall that the distance between two points is just the norm of the vector between them. Therefore, letting $\vec{v} = q(t) - p$, it suffices to test that $\vec{v} \cdot \vec{v} \le r^2$. (You could also test that $\sqrt{\vec{v} \cdot \vec{v}} \le r$. I prefer comparing squared distances to true distances because computing an extra multiplication is almost always faster than computing a square root.) If so, the line segment intersects the disk and otherwise it does not.

Solution 3:

(a) The vector $\vec{v}$ from p to q is $q - p = (q_x - p_x, q_y - p_y, q_z - p_z)$.

(b) By basic properties of the dot product, the projection of $\vec{v}$ onto $\vec{u}$ is

$$\vec{v}' \;=\; \frac{(\vec{v}\cdot\vec{u})}{(\vec{u}\cdot\vec{u})}\vec{u}.$$

The perpendicular vector comes about by subtraction $\vec{v}'' = \vec{v} - \vec{v}'$.

(c) The length of $\vec{v}'$ is

$$\|\vec{v}'\| \;=\; \sqrt{(\vec{v}'\cdot\vec{v}')} \;=\; \sqrt{{v'_x}^2 + {v'_y}^2 + {v'_y}^2}.$$

The length of $\vec{v}''$ is analogous. To determine whether q lies within the cylinder, we require that its perpendicular distance from the central axis at most r and its distance from p along the axis is at most $\ell/2$. Thus we have

$$\|\vec{v}''\| \;\leq\; r \qquad \text{and} \qquad \|\vec{v}'\| \;\leq\; \frac{\ell}{2}.$$

If so, q lies within the cylinder. Otherwise it does not.

Solution 4: There are a number of ways of deriving the answer. Assuming that it takes some time t for the ball to travel from the quarterback q to the pass receiver p, the ball travels a distance $s_p t$ and the receiver travels a distance of $s_p t$. Letting r denote the point where the ball is received, the right triangle $\triangle qpr$ has a hypotenuse of length $s_q t$ and the side opposite angle φ is of length $s_p t$. It follows from basic trigonometry that $\sin\varphi = s_p/s_q$, and therefore, $\varphi = \arcsin(ts_p/ts_q) = \arcsin(s_p/s_q)$. This is well defined if and only if $s_q > s_p$. (It is interesting to note that the distance between the quarterback and the pass receiver does not enter into this computation.)

Solution 5:

(a) Observe that a point represented relative to one joint has its x-coordinate increased when representing it relative to its parent (which is to its left). Thus, we have:

$$T_{[c\leftarrow d]} \;=\; \begin{pmatrix} 1 & 0 & 8 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[b\leftarrow c]} \;=\; \begin{pmatrix} 1 & 0 & 7 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[a\leftarrow b]} \;=\; \begin{pmatrix} 1 & 0 & 6 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

(b) Their product yields the transformation that maps a point from the tip-of-sword frame to the shoulder frame:

$$T_{[a\leftarrow d]} \;=\; T_{[a\leftarrow b]}T_{[b\leftarrow c]}T_{[c\leftarrow d]} \;=\; \begin{pmatrix} 1 & 0 & 21 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

(c) To obtain the inverse local-pose transformations simply invert these translations. The inverse of a translation by α is a translation by $-\alpha$. So, we have

$$T_{[d\leftarrow c]} \;=\; \begin{pmatrix} 1 & 0 & -8 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[c\leftarrow b]} \;=\; \begin{pmatrix} 1 & 0 & -7 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[b\leftarrow a]} \;=\; \begin{pmatrix} 1 & 0 & -6 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

(d) Let v be the position of the tip of the sword in the bind pose, let v' be its position after considering just the hand rotation, and v'' be its position after considering both the hand and elbow rotations. We will do everything relative to the shoulder frame (and to make this clear, we add the subscript $[a]$). Since, relative to the tip-of-sword frame, we have $v_{[d]} = (0, 0, 1)$, we can compute this same point relative to the shoulder frame as $v_{[a]} = T_{[a\leftarrow d]} v_{[d]}$. In particular, we have $v_{[a]} = (21, 0, 1)$.

The order of rotations is significant, and in particular, they need to be done in bottom-up order, from hand to elbow to shoulder. (See the lecture notes for an explanation of why.) In order to obtain the image v' after the hand rotation, we first convert the point into its representation relative to the hand frame (by applying the transformation $T_{[c\leftarrow a]} = T_{[c\leftarrow b]} \cdot T_{[b\leftarrow a]}$), apply the hand rotation ($\mathrm{Rot}(\theta_c)$), and then convert back to the shoulder frame (by applying $T_{[a\leftarrow c]} = T_{[a\leftarrow b]} \cdot T_{[b\leftarrow c]}$). Thus, we have

$$\begin{aligned} v'_{[a]} &= \big(T_{[a\leftarrow c]} \cdot \mathrm{Rot}(\theta_c) \cdot T_{[c\leftarrow a]}\big) \cdot v_{[a]} \\ &= \big(T_{[a\leftarrow b]} \cdot T_{[b\leftarrow c]} \cdot \mathrm{Rot}(\theta_c) \cdot T_{[c\leftarrow b]} \cdot T_{[b\leftarrow a]}\big) \cdot v_{[a]}. \end{aligned}$$

Finally, in order to obtain its image v'' after both rotations, we start with v' (which we have expressed relative to the shoulder frame), convert it into its representation relative to the elbow frame (by applying $T_{[b\leftarrow a]}$), apply the elbow rotation ($\mathrm{Rot}(\theta_b)$), and then convert back to the shoulder frame (by applying $T_{[a\leftarrow b]}$). We have

$$v''_{[a]} = \big(T_{[a\leftarrow b]} \cdot \mathrm{Rot}(\theta_b) \cdot T_{[b\leftarrow a]}\big) \cdot v'_{[a]}.$$

Combining these, we have

$$v''_{[a]} = \big(T_{[a\leftarrow b]} \cdot \mathrm{Rot}(\theta_b) \cdot T_{[b\leftarrow a]}\big)\big(T_{[a\leftarrow b]} \cdot T_{[b\leftarrow c]} \cdot \mathrm{Rot}(\theta_c) \cdot T_{[c\leftarrow b]} \cdot T_{[b\leftarrow a]}\big) \cdot v_{[a]}.$$

We can simplify this by observing that $T_{[b\leftarrow a]}$ and $T_{[a\leftarrow b]}$ are inverses, so their product cancels out. Thus, we have the following result:

$$v''_{[a]} = \big(T_{[a\leftarrow b]} \cdot \mathrm{Rot}(\theta_b) \cdot T_{[b\leftarrow c]} \cdot \mathrm{Rot}(\theta_c) \cdot T_{[c\leftarrow b]} \cdot T_{[b\leftarrow a]}\big) \cdot v_{[a]}.$$

The sequence of matrices in the parentheses is the final answer.

Solution 6:

(a) The homogeneous coordinates of p with respect to each of the frames are:

(i) Barrel frame: $p_{[c]} = (5, 0, 1)^{\mathsf{T}}$

(ii) Elbow frame: $p_{[b]} = (17, 0, 1)^{\mathsf{T}}$

(iii) Base frame: $p_{[a]} = (17, 20, 1)^{\mathsf{T}}$

(Note that these are all column vectors.)

(b) The local-pose transformation from c to b involves increasing the x-coordinate by 12 and the local-pose transformation from b to a involves increasing the y-coordinate by 20. Thus, we have

$$T_{[b\leftarrow c]} = \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[a\leftarrow b]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 20 \\ 0 & 0 & 1 \end{pmatrix}.$$

(c) By combining these two, it follows that the transformation $T_{[a\leftarrow c]}$ both increases x by 12 and increases y by 20:

$$T_{[a\leftarrow c]} \;=\; T_{[a\leftarrow b]}T_{[b\leftarrow c]} \;=\; \begin{pmatrix} 1 & 0 & 12 \\ 0 & 1 & 20 \\ 0 & 0 & 1 \end{pmatrix}$$

(d) The inverses just involve decreases by 12 and 20, respectively. So we have:

$$T_{[c\leftarrow b]} \;=\; \begin{pmatrix} 1 & 0 & -12 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[b\leftarrow a]} \;=\; \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -20 \\ 0 & 0 & 1 \end{pmatrix}.$$

(e) In order to achieve the desired rotation, we first convert $p_{[a]}$ to $p_{[c]}$ by applying the transformation $T_{[c\leftarrow a]}$. We then apply the rotation transformation $\mathrm{Rot}(\theta_c)$ at the barrel joint, then apply $T_{[b\leftarrow c]}$ to convert to the elbow joint and apply $\mathrm{Rot}(\theta_b)$, and finally apply $T_{[a\leftarrow b]}$ to convert from to the base joint and apply $\mathrm{Rot}(\theta_a)$. We are now in the coordinate frame for the base joint, so no further conversions are needed. Since all this proceeds from right to left, we have

$$p'_{[a]} \;=\; \big(\mathrm{Rot}(\theta_a)\cdot T_{[a\leftarrow b]}\cdot \mathrm{Rot}(\theta_b)\cdot T_{[b\leftarrow c]}\cdot \mathrm{Rot}(\theta_c)\cdot T_{[c\leftarrow a]}\big)p_{[a]}.$$

The final matrix is the quantity in parentheses.

Solution 7:

(a) Using the scalar-vector manner of expressing quaternions, the quaternion that yields the rotation is given by

$$\begin{aligned} \mathbf{q} \;&=\; \left(\cos\frac{\theta}{2}, \left(\sin\frac{\theta}{2}\right)\vec{u}\right) \;=\; \left(\cos 30^\circ, \sin 30^\circ \left(\frac{1}{3},\frac{2}{3},\frac{2}{3}\right)\right) \;=\; \left(\frac{\sqrt{3}}{2}, \frac{1}{2}\left(\frac{1}{3},\frac{2}{3},\frac{2}{3}\right)\right) \\ &=\; \left(\frac{\sqrt{3}}{2}, \left(\frac{1}{6},\frac{1}{3},\frac{1}{3}\right)\right). \end{aligned}$$

(As a quick sanity check, observe that if you square the components of the above vector and add them, the sum is $\frac{3}{4}+\frac{1}{36}+\frac{1}{9}+\frac{1}{9}=1$. So this is a unit quaternion, as desired.)

(b) Using the fact that $i^2=-1$ and $ij=k$, we find that the product of these two quaternions is

$$\begin{aligned} \mathbf{q}_1\cdot\mathbf{q}_2 \;&=\; (1+2i)(3i+4j) \;=\; 3i+4j+6i^2+8ij \;=\; 3i+4j-6+8k \\ &=\; -6+3i+4j+8k, \end{aligned}$$

which is the quaternion $(-6,3,4,8)$.

You could also do this using the equation given in class $\mathbf{q}_1\cdot\mathbf{q}_2 = (st-(u\cdot v), sv+tu+u\times v)$, where $\mathbf{q}_1=(s,\vec{u})=(1,(2,0,0))$ and $\mathbf{q}_2=(t,\vec{v})=(0,(3,4,0))$. In this case, we have $u\cdot v=6$ and $u\times v=(0,0,8)$, so this yields

$$(0-6,\; 1\cdot(3,4,0)+0\cdot(2,0,0)+(0,0,8)) \;=\; (-6,(3,4,8)).$$

Solution 8:

(a) The lower left corner of square $[i, j]$ is the point $(i\Delta, j\Delta)$. The corners of the square $[i, j]$ are $((i + b)\Delta, (j + c)\Delta)$, where b and c take on all possible combinations of the values 0 and 1, that is, (0,0), (0,1), (1,0), (1,1).

(b) The point (x, y) lies within the grid square $[i, j] = [\lfloor x/\Delta \rfloor, \lfloor y/\Delta \rfloor]$.

(c) This problem is remarkably tricky, and there are a number of different solutions. Note that there is an easy $O(n^2)$ solution that arises by classifying all the square of the grid, and outputting those crossed by the line.

Here is a simple $O(n)$-time solution, which assumes that a is nonnegative (it can even be greater than 1). First, we find the lowest row j (if any) crossed by the line on column $i = 0$. If every square on this column lies beneath the line, then the algorithm can safely terminate because the slope is nonnegative and so no square can be crossed. (If this occurs, the value of j will be set to n, and the second loop will never be executed.)

Otherwise, we assume the invariant that the current square $[i, j]$ is crossed by the line. Because the slope is nonnegative, the line will either go next to the square immediately to the right $[i + 1, j]$ or to the square immediately above $[i, j + 1]$. (We ignore the degenerate case where the line goes through the upper-right corner.) If the square to the immediate right is crossed, we go there next (by incrementing i). Otherwise we to the square above (by incrementing j).

```
i = j = 0;
while (j < n && classify(i, j, a, b) < 0) j++;
while (i < n && j < n) {
  output "[i, j]";
  if (classify(i+1, j, a, b) == 0) i++;
  else j++;
}
```

The running time is $O(n)$. The initial while-loop can iterate at most n times. Otherwise, with each iteration we either increment i or j, which can be done at most $2n$ times.