Solutions to Homework 1

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) Bilinearity means that the operator is a linear function of both of its arguments. In particular, for any vectors \( \vec{u}, \vec{v}, \vec{w} \) and scalar \( \alpha \),

\[
(\vec{u}, \vec{v} + \vec{w}) = (\vec{u}, \vec{v}) + (\vec{u}, \vec{w}) \quad \text{and} \quad (\vec{u}, \alpha \vec{v}) = \alpha (\vec{u}, \vec{v}).
\]

By symmetry, these are also true if we switch the order of the two operands.

(c) The associated quaternion \( q \) for this rotation is the unit quaternion \( (\cos(\theta/2), (\sin(\theta/2))\hat{u}) \), where \( \theta \) is the rotation angle and \( \hat{u} \) is the normalization of \( \vec{u} \) to unit length. Since \( \|\vec{u}\| = 3 \), we have \( \hat{u} = (1/3, 2/3, 2/3) \). Also, we have \( \cos \theta/2 = \sqrt{3}/2 \) and \( \sin \theta/2 = 1/2 \). Putting this together, we have

\[
q = \left( \frac{\sqrt{3}}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3} \right).
\]

As a check of our work, the sum of the squares of the components of \( q \) is \( 3/4 + 1/36 + 1/9 + 1/9 = (27 + 1 + 4 + 4)/36 = 1 \).

(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.

Solution 2: The zombie is facing you if, relative to your position, the sequence of the zombie’s three point \( \langle p_1, p_2, p_3 \rangle \) appear in counter-clockwise order. Therefore, the zombie is facing you if \( \text{Orient}(p_1, p_2, p_3, q) = +1 \). Otherwise, you are either to his side or behind him. This is equivalent to the following determinant inequality:

\[
\begin{vmatrix}
1 & 1 & 1 & 1 \\
p_{1,x} & p_{2,x} & p_{3,x} & q_x \\
p_{1,y} & p_{2,y} & p_{3,y} & q_y \\
p_{1,z} & p_{2,z} & p_{3,z} & q_z
\end{vmatrix} > 0.
\]
**Solution 3:** We first compute the distance from \( c \) to its closest point on the bounding box \( C_1 \). Note that the closest point on \( C_1 \) to \( c \) could be a vertex, on an edge, or in the interior of a face of \( C_1 \). There is an elegant way to distinguish among the possibilities by considering each dimension individually. Define \( \delta_x \) to be the distance from \( c_x \) to the interval \([p_x^-, p_x^+]\) (see Fig. 1 for a 2-dimensional example). In particular:

\[
\delta_x = \begin{cases} 
  p_x^- - c_x & \text{if } c_x < p_x^- \\
  0 & \text{if } p_x^- \leq c_x \leq p_x^+ \\
  c_x - p_x^+ & \text{if } p_x^+ < c_x.
\end{cases}
\]

Define \( \delta_y \) and \( \delta_z \) analogously. Then, the distance from \( c \) to \( C_1 \) is just \( \sqrt{\delta_x^2 + \delta_y^2 + \delta_z^2} \). If this quantity is less than or equal to \( r \), then \( C_1 \) and \( C_2 \) collide, otherwise they do not.

**Solution 4:**

(a) (i) Letting \( a = g/2, b = -v_{0,y}, \) and \( c = -h, \) we seek the value of \( t \) such that \( at^2 + bt + c = 0. \) (We have intentionally negated the coefficients so that \( a > 0 \).) By the quadratic formula we have

\[
t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{v_{0,y} \pm \sqrt{v_{0,y}^2 + 2gh}}{g}.
\]

Note that the quantity under the square-root sign is positive and is larger than \( v_{0,y} \), which implies that both roots exist, one is positive and one is negative. Clearly, we want the positive root, which implies that we take the “+” root from the “±” option. Therefore, we have \( t = \left( v_{0,y} + \sqrt{v_{0,y}^2 + 2gh} \right) / g \).

(ii) The value of \( \ell \) is the horizontal distance traveled, that is, \( v_{0,x}t \), where \( t \) is the value given in part (i).

(b) We solve this by generalizing the approach from part (a). The value of \( t \) depends only on the height \( h \) and the \( y \)-coordinate of the velocity \( v_{0,y} \), and so the same formula from part (a) can be applied here

\[
t = \frac{v_{0,y} \pm \sqrt{v_{0,y}^2 + 2gh}}{g}.
\]
We can split the horizontal portion of the velocity into its $x$- and $z$-components. Following part (a), the $x$-component of the distance traveled $v_{0,x}t$, and the $z$-component of the distance traveled is $v_{0,z}t$. Since it starts at the $(x,z)$-coordinates $(p_x,p_z)$, its position on the $(x,z)$-plane is $(p_x + v_{0,x}t, p_z + v_{0,z}t)$. Since it lands at $y = 0$, the final landing point is

$$q = (p_x + v_{0,x}t, 0, p_z + v_{0,z}t).$$

This is the location where to center the aiming aid.

![Figure 2: Solution to Problem 4.](image)

We could also solve the problem by reducing it to the 2-dimensional plane upon which the projectile travels. Doing so yields a velocity vector of $(\sqrt{v_{0,x}^2 + v_{0,z}^2}, v_y)$ and the distance $\ell$ that is traveled is $t\sqrt{v_{0,x}^2 + v_{0,z}^2}$. (We omit the details.)

**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) The inverse local-pose transformations simply invert these translations:

$$T_{[d \leftarrow c]} = \begin{pmatrix} 1 & 0 & -8 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad T_{[c \leftarrow b]} = \begin{pmatrix} 1 & 0 & -7 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad T_{[b \leftarrow a]} = \begin{pmatrix} 1 & 0 & -6 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. $$
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 \( v_{[a]} = T_{[a←d]} v_{[d]} \). 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 \( T_{[c←b]} = T_{[b←a]} \)), apply the hand rotation (\( \text{Rot}(θ_c) \)), and then convert back to the shoulder frame (by applying \( T_{[a←c]} = T_{[a←b]} \cdot T_{[b←a]} \)). Thus, we have

\[
v'_{[a]} = (T_{[a←c]} \cdot \text{Rot}(θ_c) \cdot T_{[c←a]} \cdot T_{[b←a]}) \cdot v_{[a]},
\]

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 \( T_{[b←a]} \)), apply the elbow rotation (\( \text{Rot}(θ_b) \)), and then convert back to the shoulder frame (by applying \( T_{[a←b]} \)). We have

\[
v''_{[a]} = (T_{[a←b]} \cdot \text{Rot}(θ_b) \cdot T_{[b←a]} \cdot T_{[b←c]} \cdot \text{Rot}(θ_c) \cdot T_{[c←b]} \cdot T_{[b←a]}) \cdot v_{[a]}.
\]

Combining these, we have

\[
v''_{[a]} = (T_{[a←b]} \cdot \text{Rot}(θ_b) \cdot T_{[b←a]} \cdot T_{[a←c]} \cdot \text{Rot}(θ_c) \cdot T_{[c←b]} \cdot T_{[b←a]}) \cdot v_{[a]},
\]

We can simplify this a bit by observing that \( T_{[b←a]} \) and \( T_{[a←b]} \) are inverses, so their product cancels out.

\[
v''_{[a]} = (T_{[a←b]} \cdot \text{Rot}(θ_b) \cdot T_{[b←c]} \cdot \text{Rot}(θ_c) \cdot T_{[c←b]} \cdot T_{[b←a]}) \cdot v_{[a]},
\]

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