Solutions to Homework 1

Solution 1:

(a) **Collider:** (i) Attached to a wall, preventing objects from passing through it. (ii) Attached to the player to signal when it has been hit by a projectile.

Trigger: (i) Attached to a doorway, signaling whenever some game object walks into or out of the room. (ii) Associated with a large ball enclosing an NPC, signaling that a threat is approaching and that action is needed (fight or flight).

(b) Let $m = \frac{1}{2}b + \frac{1}{2}c$ be the midpoint of $\overline{bc}$. For $0 \le i \le 4$, let $\alpha_i \leftarrow i/4$. Then $p[i] = (1-\alpha_i)a + \alpha_i m$.

(c) With each update call we rotate (in degrees) by $(90/4) \cdot \mathsf{Time.deltaTime} = 22.5 \cdot \mathsf{Time.deltaTime}$. For example, if the rotation had been about the y-axis (Vector3.up), then we could do this with the Unity command

```
transform.Rotate (Vector3.up * 22.5f * Time.deltaTime);
```

Solution 2:

(a) $\vec{u} = q - p = (q_x - p_x, q_y - p_y, q_z - p_z)$, and $\vec{v} = c - p = (c_x - p_x, c_y - p_y, c_z - p_z)$.

(b) We can obtain t as follows. Let $\vec{w}$ be the orthogonal projection of $\vec{v}$ onto $\vec{u}$. That is, $\vec{w} \leftarrow \alpha\vec{u}$, where $(\vec{u} \cdot \vec{v})/(\vec{u} \cdot \vec{u})$. Then $t = p + \vec{w}$.

(c) Consider α from (b). Observe that α is a stretch value, which is 0 when c projects onto p and is 1 when c projects onto q. If $0 \le \alpha \le 1$, then t lies within the line segment $\overline{pq}$. If $\alpha < 0$ or $\alpha > 1$, then t lies outside this segment, being beyond p if $\alpha < 0$ and beyond q if $\alpha > 1$. Let $\mathrm{clamp}(x, a, b)$ be a function that clamps the scalar x to the interval $[a, b]$.

$$\alpha' \leftarrow \mathrm{clamp}(\alpha, 0, 1), \qquad t' \leftarrow (1 - \alpha')p + \alpha' q.$$

We could express t' equivalently as $p + \alpha'\vec{u}$, which is a bit simpler to compute.

(d) The two colliders intersect if and only if the distance between c and t' is at most $r + s$:

$$\|t' - c\| = \sqrt{(t'_x - c_x)^2 + (t'_y - c_y)^2 + (t'_y - c_y)^2} \le r + s.$$

(e) In the code below the variable `alpha` takes the role of both α and α', and variable `t` takes the role of t' (since it is based on the clamped value of α). The code below is quite clean since it does not need reference individual coordinates.

```
bool SphereCapsuleCollide(Vector3 c, float r, Vector3 p, Vector3 q, float s) {
  Vector3 u = q - p;                                         // vector from p to q
  Vector3 v = c - p;                                         // vector from p to c
  float alpha = Vector3.Dot(u, v) / Vector3.Dot(u, u);        // projection factor
  Mathf.Clamp(alpha, 0, 1);                                 // clamp alpha to [0, 1]
  Vector3 t = p + u * alpha;                  // t is the closest point to c along pq
  return Vector3.Distance(c, t) <= r + s;                 // close enough to collide?
}
```

Solution 3: As mentioned in the problem description, the player's frames origin is p. Here is how to compute the other basis vectors.

(a) (For this part, we assume the *right-hand rule*. For a left-handed system, reverse the arguments to cross products.) First, the player's up-vector $\vec{u}$ is just the normalization of the vector from c through p, that is

$$\vec{u} \leftarrow \text{normalize}(p - c) = \frac{p - c}{\|p - c\|}.$$

(Recall that the length of a vector $\vec{v}$ can be computed as $\|\vec{v}\| \leftarrow \sqrt{\vec{v} \cdot \vec{v}}$.)

Next, to compute the player's right-vector $\vec{r}$, we observe that it must be perpendicular to the equatorial plane containing p and q, or equivalently, it must be perpendicular to both of the up-vector $\vec{u}$ and $\vec{w} = q - c$. Using the standard right-handed cross product, we have

$$\vec{r} \leftarrow \text{normalize}(\vec{w} \times \vec{u}), \quad \text{where } \vec{w} \leftarrow q - c.$$

Finally, to compute the forward vector $\vec{f}$, we observe that this vector is perpendicular to both $\vec{u}$ and $\vec{r}$. Given the relative orientations of the vectors, we have

$$\vec{f} \leftarrow \vec{u} \times \vec{r}.$$

Note that there is no need to perform a normalization here. Since $\vec{u}$ and $\vec{r}$ are already of unit length and perpendicular to each other, it follows that the above cross product returns a vector of unit length.

(b) (For this part, we assume the *left-hand rule* as does Unity. For a right-handed system, reverse the arguments to cross products.) The code below follows the general structure of the above methods. We assume that p, c, and q are not collinear.

```
void PlayerFrame(Vector3 c, float r, Vector3 p, Vector3 q,
                     out Vector3 u, out Vector3 f, out Vector3 r) {
    u = (p - c).normalized;                                          // up vector
    Vector3 w = q - c;                      // another vector on the equatorial plane
    r = Vector3.Cross(u, w).normalized;                                // right vector
    f = Vector3.Cross(r, u);                                           // front vector
}
```

Solution 4:

(a) The boundary of S_0 can be covered by 4 circular disks of diameter 1. At each new stage, each edge of length x in the previous shape is replaced by 4 edges each of length $\sqrt{2}/4$. Let $\varepsilon_i = (\sqrt{2}/4)^i$. It follows that we can cover the boundary of S_i with $4 \cdot 4^i = 4^{i+1}$ disks of radius ε_i. Thus, the fractal dimension is

$$\lim_{i\to\infty} \frac{\ln N(S_i, \varepsilon^i)}{\ln 1/\varepsilon_i} = \lim_{i\to\infty} \frac{\ln 4^{i+1}}{\ln(4/\sqrt{2})^i} = \lim_{i\to\infty} \frac{i+1}{i} \frac{\ln 4}{\ln(4/\sqrt{2})} = \frac{\ln 4}{\ln(4/\sqrt{2})} = \frac{4}{3}.$$

We are using the fact that $\lim_{i\to\infty}(i+1)/i = 1$. The final simplification to 4/3 is not required, but it can be derived by expressing everything as a power of 2. That is, $\ln 4/\ln(4/\sqrt{2}) = \ln(2^2)/\ln(2^2/2^{1/2}) = 2\ln 2/(3/2)\ln 2 = 2/(3/2) = 4/3$.

(b) (i) The initial step size is the length of the unit square, $d = 1$. With each new iteration, the length decreases by a factor of $\sqrt{2}/4$, and therefore $d_i = (\sqrt{2}/4)^i$. Although the turns are multiples of 90°, the starting heading is a multiple of 45°, and so we will set the angular displacement to $\delta = 45°$. (See below for an alternative solution.)

(ii) The variables V of the system consist of $\{\mathsf{F}, +, -\}$, which have the standard turtle-geometry interpretations.

(iii) The starting string ω draws the unit square. Since $\delta = 45°$, we need two turns at each corner. Thus we have $\omega = \mathsf{F}++\mathsf{F}++\mathsf{F}++\mathsf{F}$. (We could add an extra ++ to the end so that the turn angles sum to exactly 360°.)

(iv) We start by making a 45° clockwise turn ($-$), then draw a segment (F), then a 90° counterclockwise turn ($++$), then draw two segments (FF), then a 90° clockwise turn ($--$), and then end with one more segment (F).

There is an issue, however. We have made one more clockwise than counterclockwise turn, and hence we do not end in the same direction that we started. This is an issue because we have concatenated these sequences together under the assumption that we end each sequence in the same direction that it started. To correct this we add a final counterclockwise turn ($+$). Thus we have the following single production rule:

$$p = \{\mathsf{F} \to -\mathsf{F}++\mathsf{F}\,\mathsf{F}--\mathsf{F}+\}.$$

Alternate solution: Here is an arguably simpler way to fix the heading issue. Observe that the initial heading rotates 45° clockwise with each level. Thus, for S_i we define the initial state of the turtle to be $(0, 0, -i \cdot 45°)$. Now, since the turtle starts off pointing the proper direction, we can omit the initial "$-$", and (because we are balanced) we can also omit the final "+". If we do this, then all the turns are $\pm 90°$, and so we can define $\delta = 90°$. This means that we can replace the "++" and "$--$" above with singles "+" and "$-$". This leads to the following simpler production rule:

$$p = \{\mathsf{F} \to \mathsf{F}+\mathsf{F}\,\mathsf{F}-\mathsf{F}\},$$

and $\omega = \mathsf{F}+\mathsf{F}+\mathsf{F}+\mathsf{F}$.

Solution 5:

(a) In going from frame d to c, a point's y-coordinate decreases by D_0 units, in going from c to b, a point's x-coordinate increases by W_0 units, and in going from b to a, a point's y-coordinate increases by H_0 units. Therefore, we have

$$T_{[c \leftarrow d]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -70 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[b \leftarrow c]} = \begin{pmatrix} 1 & 0 & 100 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{[a \leftarrow b]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 130 \\ 0 & 0 & 1 \end{pmatrix}.$$

(b) By multiplying these matrices together, we obtain a matrix

$$T_{[a \leftarrow d]} = \begin{pmatrix} 1 & 0 & 100 \\ 0 & 1 & 60 \\ 0 & 0 & 1 \end{pmatrix}.$$

As a check of correctness, we see that

$$T_{[a \leftarrow d]} p_{[d]} = \begin{pmatrix} 1 & 0 & 100 \\ 0 & 1 & 60 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ -20 \\ 1 \end{pmatrix} = \begin{pmatrix} 100 \\ 40 \\ 1 \end{pmatrix} = p_{[a]},$$

as desired.