CS52: Homework 4 Solutions

Problem 1: Keyframed Animation

(a) q_{4k} should be changed for the position, and q_{4k+1} should be changed for the velocity.

(b) The choice is a good one. Consider the elbow of your arm, as the object position at time t_k. Obviously, you might not wanna enforce the continuity.

Problem 2: Deformations

(a)

$$P_d = (P_x \cos(P_y * \theta), P_y, -P_z sin(P_y * \theta))$$

(b)

$$P' = P * ((1-y) * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + y * \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix})$$

(c) the skinning algorithm tends to decrease the quality for larger rotations. The averaging process applied to the vertices might cause the shrink and lost of volume. Servere bend and twist angles could cause collapse of joints too.

Problem 3: Texturing

(a)this is helpful. Since we know the distance between a ray and the rays for a neighboring pixel, by using this algorithm, we could efficiently filter the texture. The only thing, that we should keep in mind, is to get the index for each level of textures, for each 3d points.

(b) transform the point P into the light-world, and get P'(using matrix M). calculate the corresponding image point (x',y',z')for P'. look up the depth value in the shadow map, for

point (x',y'). if the depth value in the shadow map is smaller than z', P is in shadow. otherwise, P is not in shadow.

(c) If we try to update the original textures and create new ones, the UV parameterization of the surface need not to be changed, and modifying the texture will introduce no overhead in rendering stage. However, this approach requires the game program to maintain a copy of texture for each car. which will be a waste of storage when, sometimes, cars can share surface textures and scratch patterns. Moreover, if the car surfaces will change rapidly at run time, updating textures frequently will require large data throughput to the texture buffer in graphics chip, which will reduce the performance of the game.

If we try to use new UV sets and textures to represent the scratch patterns, the UV parameterization of the surface need to be update and more texture mapping need to be done. This approach allow cars sharing surface texture and scratch patterns. Furthermore, the surface update will be trivial, since only the UV parameters of a few vertices need to be change. However, this approach requires runtime texture blending in the graphics pipeline(usually the fragment shader) and reduce the rendering performance of the game.

Problem 4: Comparing Raytracing to the Graphcis Pipelines

(a)

- Image with no lights
  - Graphic Pipeline: o; the scene needs to be rendered only once.
  - Basic Raytracer: $nO(o)$; intersection test for each ray.
  - Advanced Raytracer: $nO(ln(o))$.
- Image with shadow
  - Graphic Pipeline: $(l+1)o$; First, a shadow map need to be generated for each light, that is l passes. Then, render the final image once the shadow maps are generated.
  - Basic Raytracer: $n(l+1)O(o)$; First, each ray will intersect with the scene to get intersection point. Then, l shadow rays will be generated to intersect with the scene to generate shadow.
  - Advanced Raytracer: $n(l+1)O(ln(o))$
- Image with shadow and reflection
  - Graphic Pipeline: $(l+1)o+6r(o-1)$; First, a shadow map need to be generated for each light, that is l passes. Second, for each object has reflection, render the scene with without that object 6 times to generate a environment cube mapping. Then, render the final image.

- Basic Raytracer: $n(l+1+r/o)O(o)$; First, each ray will intersect with the scene to get intersection point. Then, l shadow rays will be generated to intersect with the scene to generate shadow. For rays that hit objects with reflection, a reflection ray will be generated to intersect with the scene.
- Advanced Raytracer: $n(l+1+r/o)O(ln(o))$.

(b) For pipeline, the number of lighting computations is $a \times n$. For raytracer, it is n.

(c)The graphics pipeline is slow. But it is still widely used. This is because it is well suited to the rendering process. It allows GPU to function as a stream processor since all vertices and fragments can be thought of as independent. This allows all stages of the pipeline to be used simultaneously for different vertices or fragments as they work their way through the pipe. This independence also allows graphics processors to use parallel processing units to process multiple vertices or fragments in a single stage of the pipeline at the same time.

Extra credit: shadow computations

(a) The shadows are prone to aliasing when zooming into the shadow boundaries. Actually, the accuracy of a shadow map is limited by its resolution.

(b) For simple scenes, we could use the raytraced shadows, because that is more accurate than the shadows computed by shadow mapping algorithms, while for larger and more complex scenes, we should use shadow maps because they are much faster than the raytraced shadow algorithms.