Компьютерная графика

Практика 10: Скелетная анимация

2021

Задание 1

Вычисляем позу для модели и применяем её

- В массиве `poses` 6 фиксированных поз, преобразования костей в них заданы относительно родительских костей!

Задание 1

Вычисляем позу для модели и применяем её

- В массиве `poses` 6 фиксированных поз, преобразования костей в них заданы относительно родительских костей!
- Заводим массив объектов `bone_pose` для хранения посчитанных преобразований скелета (по одному преобразованию на кость)

Задание 1

Вычисляем позу для модели и применяем её

- В массиве `poses` 6 фиксированных поз, преобразования костей в них заданы относительно родительских костей!
- Заводим массив объектов `bone_pose` для хранения посчитанных преобразований скелета (по одному преобразованию на кость)
- Вычисляем преобразования:
  - Для корневой кости (`parent == -1`) копируем преобразование из выбранной позы (например, `poses[0]`)
  - Для некорневой кости берём преобразование из выбранной позы и умножаем *слева* на уже посчитанное преобразование родительской кости
  - N.B.: в данных кости идут в порядке топологической сортировки, т.е. родительская кость всегда идёт раньше дочерней

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- Заводим по массиву uniform-переменных на повороты, сдвиги и масштабирования костей (всего 3 массива uniform'ов, размером по 64 элемента)

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- Заводим по массиву uniform-переменных на повороты, сдвиги и масштабирования костей (всего 3 массива uniform'ов, размером по 64 элемента)
- Заводим три массива под uniform location и заполняем их (`glGetUniformLocation`, для генерации имён можно воспользоваться конструкцией `"bone_rotation[" + std::to_string(i) + "]"`)

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- Заводим по массиву uniform-переменных на повороты, сдвиги и масштабирования костей (всего 3 массива uniform'ов, размером по 64 элемента)
- Заводим три массива под uniform location и заполняем их (`glGetUniformLocation`, для генерации имён можно воспользоваться конструкцией `"bone_rotation[" + std::to_string(i) + "]"`)
- При рендеринге выставляем зачения всех uniform-переменных для костей из посчитанного (на предыдущем слайде) массива преобразований

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- Заводим по массиву uniform-переменных на повороты, сдвиги и масштабирования костей (всего 3 массива uniform'ов, размером по 64 элемента)
- Заводим три массива под uniform location и заполняем их (`glGetUniformLocation`, для генерации имён можно воспользоваться конструкцией `"bone_rotation[" + std::to_string(i) + "]"`)
- При рендеринге выставляем зачения всех uniform-переменных для костей из посчитанного (на предыдущем слайде) массива преобразований
- N.B.: координаты кватернионов идут в порядке (w, x, y, z)

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- В вершинном шейдере применяем к позиции входной вершины преобразования двух костей (их индексы лежат в `in_bone_id`) и суммируем с весами из `in_bone_weight`

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- В вершинном шейдере применяем к позиции входной вершины преобразования двух костей (их индексы лежат в `in_bone_id`) и суммируем с весами из `in_bone_weight`
- Делаем то же самое с нормалями (к ним не нужно применять translation!)

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- В вершинном шейдере применяем к позиции входной вершины преобразования двух костей (их индексы лежат в `in_bone_id`) и суммируем с весами из `in_bone_weight`
- Делаем то же самое с нормалями (к ним не нужно применять translation!)
- N.B.: код для вращения кватернионами в шейдере уже есть

Задание 1 (продолжение)

Вычисляем позу для модели и применяем её

- В вершинном шейдере применяем к позиции входной вершины преобразования двух костей (их индексы лежат в `in_bone_id`) и суммируем с весами из `in_bone_weight`
- Делаем то же самое с нормалями (к ним не нужно применять translation!)
- N.B.: код для вращения кватернионами в шейдере уже есть
- N.B.: не забываем про матрицу model: её всё ещё нужно применить после скелетных преобразований

Задание 2

Интерполируем между соседними позами

- Как-нибудь вычисляем номер текущей позы в зависимости от времени (например, `floor(time)`)
- Номер следующей позы = (номер текущей позы + 1)

Задание 2

Интерполируем между соседними позами

- Как-нибудь вычисляем номер текущей позы в зависимости от времени (например, `floor(time)`)
- Номер следующей позы = (номер текущей позы + 1)
- Вычисляем параметр интерполяции между текущей позой и следующей (например, `t = time - floor(time)`)

Задание 2

Интерполируем между соседними позами

- Как-нибудь вычисляем номер текущей позы в зависимости от времени (например, `floor(time)`)
- Номер следующей позы = (номер текущей позы + 1)
- Вычисляем параметр интерполяции между текущей позой и следующей (например, `t = time - floor(time)`)
- При вычислении массива преобразований из задания 1 не просто берём преобразование из позы, а интерполируем две соседние позы

Задание 2

Интерполируем между соседними позами

- Как-нибудь вычисляем номер текущей позы в зависимости от времени (например, `floor(time)`)
- Номер следующей позы = (номер текущей позы + 1)
- Вычисляем параметр интерполяции между текущей позой и следующей (например, `t = time - floor(time)`)
- При вычислении массива преобразований из задания 1 не просто берём преобразование из позы, а интерполируем две соседние позы
- N.B.: для интерполяции кватернионов можно воспользоваться функцией `glm::slerp`
- N.B.: для интерполяции масштабов и сдвигов можно воспользоваться функцией `glm::mix`

Задание 3

Добавляем easing

- Применяем к параметру интерполяции какую-нибудь easing-функцию, например $3t^2 - 2t^3$