

Компьютерная графика

Практика 13: Скелетная анимация

2022

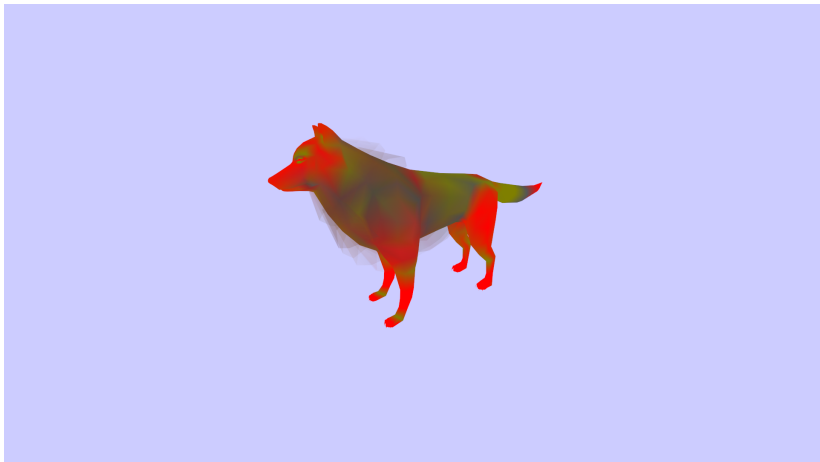


Задание 1

Проверяем, что нормально заданы веса вершин

- ▶ В вершинном шейдере добавляем два новых атрибута вершин:
 - ▶ Номера костей для вершины: `ivec4 in_joints,`
`location = 3`
 - ▶ Веса костей для вершины: `vec4 in_weights,`
`location = 4`
 - ▶ (в VAO эти атрибуты уже настроены)
- ▶ Передаём `in_weights` во фрагментный шейдер и используем в качестве цвета

Задание 1



Задание 2

Применяем преобразования костей

- ▶ Возвращаем нормальное вычисление цвета во фрагментном шейдере
- ▶ В вершинном шейдере: заводим uniform-массив для матриц костей
`uniform mat4x3 bones[64]`
 - ▶ Location получаем как `glGetUniformLocation(program, "bones")`
- ▶ Вычисляем взвешенное среднее `mat4x3 average` матриц для вершины (`in_joints` – индексы четырёх матриц, `in_weights` – их веса)
- ▶ Перед применением матрицы `model` применяем к входной вершине матрицу `mat4(average)`
- ▶ Перед применением матрицы `mat3(model)` применяем к входной нормали матрицу `mat3(average)`
- ▶ В цикле рендеринга заводим переменную `scale` как-то меняющуюся со временем (например, `0.75 + cos(time) * 0.25`)
- ▶ В цикле рендеринга заводим массив матриц `std::vector<glm::mat4x3>` размера `input_model.bones.size()` и заполняем значением `glm::mat4x3(scale)`
- ▶ Передаём матрицы в uniform-массив одним вызовом `glUniformMatrix4x3fv`

Задание 2



Задание 3

Вычисляем преобразования костей

- ▶ Достаём из входной модели анимацию с названием "01_Run"
- ▶ Для каждой кости i вычисляем её преобразование:

```
glm::mat4 transform = translation * rotation * scale
```

 - ▶ `translation` берётся из `animation.bones[i].translation(0.f)` (0 – время кадра, пока про него не думаем), остальные компоненты аналогично
 - ▶ Получить по `translation` матрицу можно через `glm::translate(glm::mat4(1.f), translation)`, аналогично `glm::scale` для `scale`
 - ▶ Получить по кватерниону `rotation` матрицу можно через `glm::toMat4`
 - ▶ Если у кости есть родитель `input_model.bones[i].parent`, домножаем матрицу `transform` на матрицу родителя:
`parent_transform * transform`
 - ▶ Записываем `transform` в `bones[i]` (в этот момент `mat4` конвертируется в `mat4x3`)
- ▶ После этого отдельным циклом домножаем каждую кость на её `inverse-bind` матрицу:

```
bones[i] = bones[i] * input_model.bones[i].inverse_bind_matrix
```

Задание 3



Задание 4

Анимлируем модель

- ▶ Вместо значения 0, передаваемого в `animation.bones[i].translation(...)` и т.п., передаём значение `std::fmod(time, animation.max_time)`

Задание 4



Задание 5*

Интерполируем между двумя анимациями

- ▶ Заводим некий параметр интерполяции, который
 - ▶ Линейно растёт ($\dots += \text{speed} * dt$), если нажата какая-то клавиша (например, `SDLK_LSHIFT` – левый shift)
 - ▶ Линейно убывает ($\dots -= \text{speed} * dt$), если эта клавиша не нажата
 - ▶ После обновления параметр ограничивается диапазоном $[0, 1]$
- ▶ Используем этот параметр для интерполяции между анимациями "01_Run" и "02_walk":
 - ▶ Для `translation` каждой кости интерполируем (`glm::lerp`) между `walk_animation.bones[i].translation` и `run_animation.bones[i].translation`
 - ▶ Аналогично для `rotation` и `scale` (для вращения лучше использовать `glm::slerp`)
- ▶ N.B. `max_time` у этих двух анимаций разный!

Задание 5*

