

Компьютерная графика

Лекция 13: Анимации, easing functions, keyframes, bitmap-анимации, кватернионы, иерархии объектов, скелетная анимация, форматы 3D моделей

2023

- Анимация (в общем смысле) – что угодно, меняющееся со временем

- Анимация (в общем смысле) – что угодно, меняющееся со временем
 - Двигающийся объект

- Анимация (в общем смысле) – что угодно, меняющееся со временем
 - Двигающийся объект
 - Анимированный элемент интерфейса

- Анимация (в общем смысле) – что угодно, меняющееся со временем
 - Двигающийся объект
 - Анимированный элемент интерфейса
 - Анимированная модель

- Анимация (в общем смысле) – что угодно, меняющееся со временем
 - Двигающийся объект
 - Анимированный элемент интерфейса
 - Анимированная модель
 - Движущаяся камера

- Анимация (в общем смысле) – что угодно, меняющееся со временем
 - Двигающийся объект
 - Анимированный элемент интерфейса
 - Анимированная модель
 - Движущаяся камера
 - etc.

- Анимация (в общем смысле) – что угодно, меняющееся со временем
 - Двигающийся объект
 - Анимированный элемент интерфейса
 - Анимированная модель
 - Движущаяся камера
 - etc.
- Сводится к вопросу о том, что и как мы меняем в зависимости от времени

- Зачем нужна анимация?

- Зачем нужна анимация?
 - Часто в ней заключается суть (напр. почти все игры)

- Зачем нужна анимация?
 - Часто в ней заключается суть (напр. почти все игры)
 - Анимация выглядит приятнее и понятнее, чем дискретное изменение состояния (напр. в UI)

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)

Анимация: общие идеи

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)
 - Будет лучше работать при выключенном VSync

Анимация: общие идеи

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)
 - Будет лучше работать при выключенном VSync
 - Будет лучше работать, когда CPU/GPU не справляются с нагрузкой

Анимация: общие идеи

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)
 - Будет лучше работать при выключенном VSync
 - Будет лучше работать, когда CPU/GPU не справляются с нагрузкой
- В общем случае анимация – это зависимость какой-то величины от времени $x = f(t)$

Анимация: общие идеи

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)
 - Будет лучше работать при выключенном VSync
 - Будет лучше работать, когда CPU/GPU не справляются с нагрузкой
- В общем случае анимация – это зависимость какой-то величины от времени $x = f(t)$
- Как задать функцию $f(t)$?

Анимация: общие идеи

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)
 - Будет лучше работать при выключенном VSync
 - Будет лучше работать, когда CPU/GPU не справляются с нагрузкой
- В общем случае анимация – это зависимость какой-то величины от времени $x = f(t)$
- Как задать функцию $f(t)$?
 - Явной формулой

Анимация: общие идеи

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)
 - Будет лучше работать при выключенном VSync
 - Будет лучше работать, когда CPU/GPU не справляются с нагрузкой
- В общем случае анимация – это зависимость какой-то величины от времени $x = f(t)$
- Как задать функцию $f(t)$?
 - Явной формулой
 - Склеить из кусочков (сплайн)

Анимация: общие идеи

- Вычисление анимаций лучше привязывать к реальному времени, а не к номеру кадра (напр. `state += speed * dt` а не `state += speed`)
 - Будет лучше работать при выключенном VSync
 - Будет лучше работать, когда CPU/GPU не справляются с нагрузкой
- В общем случае анимация – это зависимость какой-то величины от времени $x = f(t)$
- Как задать функцию $f(t)$?
 - Явной формулой
 - Склеить из кусочков (сплайн)
 - Вычислять неявно на основе текущего состояния

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на `new_x`

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 1:** запомнить старое и новое значения, интерполировать между ними учитывая прошедшее время
 $x = \text{lerp}(\text{old_x}, \text{new_x}, \text{time})$

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 1:** запомнить старое и новое значения, интерполировать между ними учитывая прошедшее время
 $x = \text{lerp}(\text{old_x}, \text{new_x}, \text{time})$
 - + Легко настраивать форму интерполяции (*easing functions*)

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 1:** запомнить старое и новое значения, интерполировать между ними учитывая прошедшее время
 $x = \text{lerp}(\text{old_x}, \text{new_x}, \text{time})$
 - + Легко настраивать форму интерполяции (*easing functions*)
 - — Нужно обрабатывать ситуацию, когда новое значение изменилось в процессе анимации

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на `new_x`

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 2:** обновлять x *только* на основе нового значения:
 $x = \text{lerp}(x, new_x, speed * dt)$
 - Если $dt > 1/speed$, возникнет нестабильность

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 2:** обновлять x *только* на основе нового значения:
 $x = \text{lerp}(x, new_x, speed * dt)$
 - Если $dt > 1/speed$, возникнет нестабильность
 - Более точная, стабильная формула:
 $x = \text{lerp}(x, new_x, \exp(- speed * dt))$

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 2:** обновлять x *только* на основе нового значения:

$x = \text{lerp}(x, new_x, speed * dt)$

- Если $dt > 1/speed$, возникнет нестабильность
- Более точная, стабильная формула:

$x = \text{lerp}(x, new_x, \exp(-speed * dt))$

- Соответствует численному решению уравнения

$$\frac{dx}{dt} = speed \cdot (x_{new} - x)$$

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 2:** обновлять x *только* на основе нового значения:
 $x = \text{lerp}(x, new_x, speed * dt)$
 - Если $dt > 1/speed$, возникнет нестабильность
 - Более точная, стабильная формула:
 $x = \text{lerp}(x, new_x, \exp(-speed * dt))$
 - Соответствует численному решению уравнения
 $\frac{dx}{dt} = speed \cdot (x_{new} - x)$
 - Удобно для анимации камеры, элементов интерфейса

Анимация одной величины

- Пусть у нас есть некая величина x , и мы хотим плавно поменять её значение на new_x
- **Вариант 2:** обновлять x *только* на основе нового значения:
 $x = \text{lerp}(x, new_x, speed * dt)$
 - Если $dt > 1/speed$, возникнет нестабильность
 - Более точная, стабильная формула:
 $x = \text{lerp}(x, new_x, \exp(-speed * dt))$
 - Соответствует численному решению уравнения
 $\frac{dx}{dt} = speed \cdot (x_{new} - x)$
 - Удобно для анимации камеры, элементов интерфейса
 - Подробная статья про этот метод

Анимации: easing functions

- При временной интерполяции между двумя значениями вместо линейной интерполяции $x = \text{lerp}(\text{old_x}, \text{new_x}, t)$ можно использовать т.н. easing functions

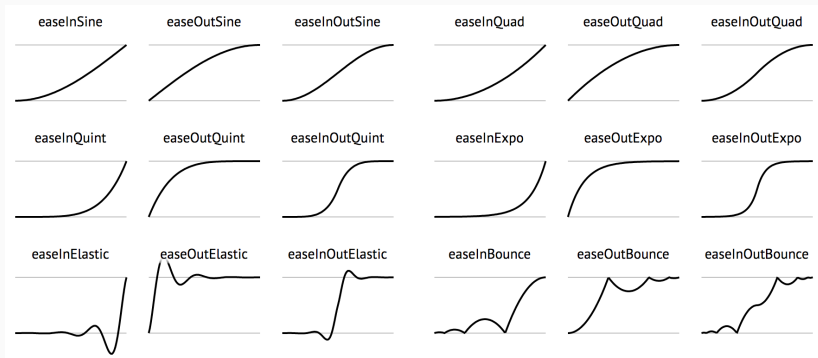
Анимации: easing functions

- При временной интерполяции между двумя значениями вместо линейной интерполяции $x = \text{lerp}(\text{old_x}, \text{new_x}, t)$ можно использовать т.н. easing functions
- Применяются к параметру интерполяции $t \in [0, 1]$ и сглаживают анимацию: $x = \text{lerp}(\text{old_x}, \text{new_x}, \text{easing}(t))$

Анимации: easing functions

- При временной интерполяции между двумя значениями вместо линейной интерполяции $x = \text{lerp}(\text{old_x}, \text{new_x}, t)$ можно использовать т.н. easing functions
- Применяются к параметру интерполяции $t \in [0, 1]$ и сглаживают анимацию: $x = \text{lerp}(\text{old_x}, \text{new_x}, \text{easing}(t))$
- Примеры easing functions:
 - $f(t) = t$
 - $f(t) = 3t^2 - 2t^3$
 - $f(t) = t^2$
 - $f(t) = 1 - (1 - t)^2$
 - $f(t) = \sqrt{t}$
 - Больше примеров: easings.net

Анимация: easing functions



Анимация: сплайны

- Часто значения интерполируют, используя сплайны: кривые, позволяющие удобно настраивать зависимость некой величины от параметра t

Анимация: сплайны

- Часто значения интерполируют, используя сплайны: кривые, позволяющие удобно настраивать зависимость некой величины от параметра t
- Обычно сплайн строится по набору значений в точках (keyframes) и, возможно, значений производных в этих точках

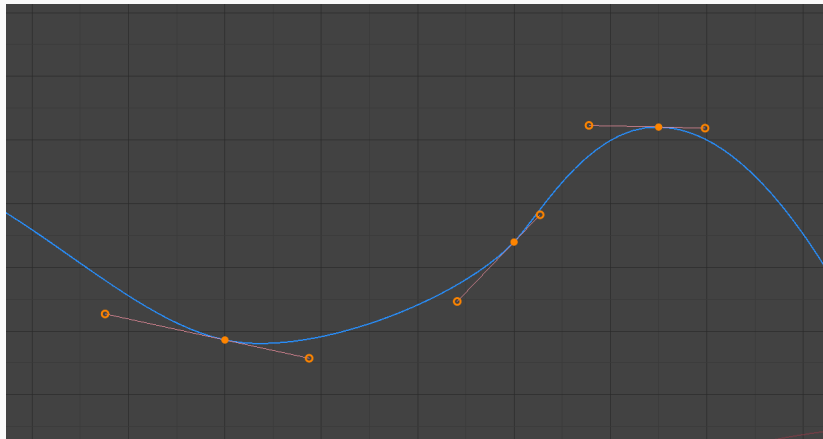
Анимация: сплайны

- Часто значения интерполируют, используя сплайны: кривые, позволяющие удобно настраивать зависимость некой величины от параметра t
- Обычно сплайн строится по набору значений в точках (keyframes) и, возможно, значений производных в этих точках
- Виды сплайнов:
 - Сплайны Безье
 - Кубические сплайны
 - B-сплайны
 - NURBS
 - etc.

Анимация: сплайны

- Часто значения интерполируют, используя сплайны: кривые, позволяющие удобно настраивать зависимость некой величины от параметра t
- Обычно сплайн строится по набору значений в точках (keyframes) и, возможно, значений производных в этих точках
- Виды сплайнов:
 - Сплайны Безье
 - Кубические сплайны
 - B-сплайны
 - NURBS
 - etc.
- **NB:** спектр применения сплайнов не ограничивается анимациями!
 - Curve fitting
 - Представление сложных геометрических форм (напр. зданий, шрифтов)
 - etc.

Анимация: сплайны



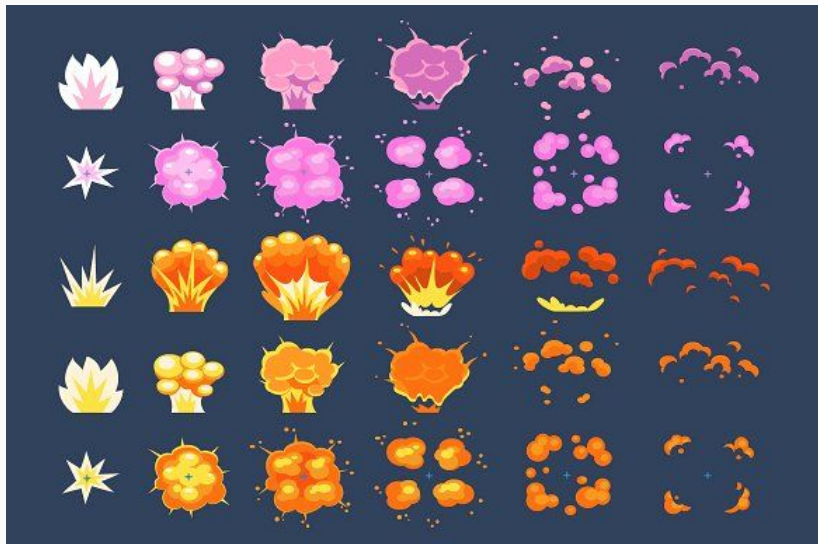
- Меняющееся со временем изображение

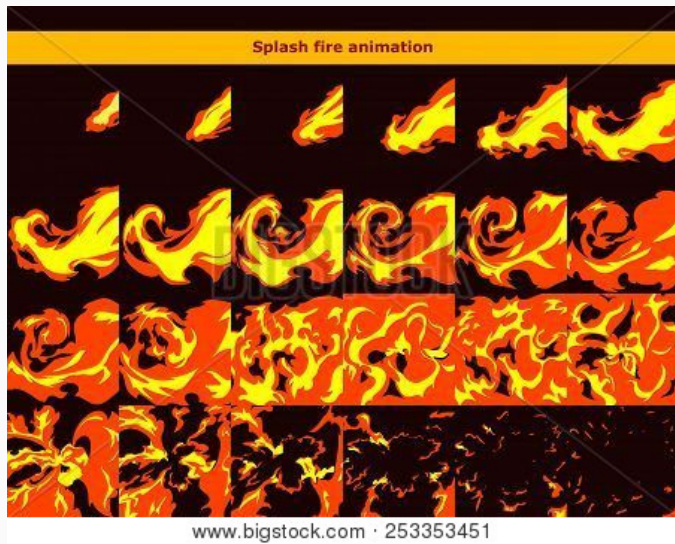
- Меняющееся со временем изображение
- 3D текстура
 - Номер кадра – 3я текстурная координата (нормированная)
 - Сама интерполирует между кадрами

- Меняющееся со временем изображение
- 3D текстура
 - Номер кадра – 3я текстурная координата (нормированная)
 - Сама интерполирует между кадрами
- 2D array текстура
 - Номер кадра – 3я текстурная координата (**не** нормированная)
 - **Не** интерполирует сама между кадрами

- Меняющееся со временем изображение
- 3D текстура
 - Номер кадра – 3я текстурная координата (нормированная)
 - Сама интерполирует между кадрами
- 2D array текстура
 - Номер кадра – 3я текстурная координата (**не** нормированная)
 - **Не** интерполирует сама между кадрами
- 2D текстурный атлас – текстура, хранящая несколько изображений бок о бок
 - По номеру кадра вычисляются настоящие текстурные координаты
 - **Не** интерполирует сам между кадрами

Текстура-атлас с анимацией





Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много

Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много
 - Применить матрицу к вектору – минимум операций сложения и умножения

Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много
 - Применить матрицу к вектору – минимум операций сложения и умножения
 - Композиция вращений – произведение матриц, довольно быстрая операция

Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много
 - Применить матрицу к вектору – минимум операций сложения и умножения
 - Композиция вращений – произведение матриц, довольно быстрая операция
 - Интерполяция между двумя матрицами вращения – почти всегда **не** матрица вращения

Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много
 - Применить матрицу к вектору – минимум операций сложения и умножения
 - Композиция вращений – произведение матриц, довольно быстрая операция
 - Интерполяция между двумя матрицами вращения – почти всегда **не** матрица вращения
- Вращения образуют 3х-мерную группу $SO(3)$, т.е. описываются 3-мя параметрами, например углами Эйлера

Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много
 - Применить матрицу к вектору – минимум операций сложения и умножения
 - Композиция вращений – произведение матриц, довольно быстрая операция
 - Интерполяция между двумя матрицами вращения – почти всегда **не** матрица вращения
- Вращения образуют 3х-мерную группу $SO(3)$, т.е. описываются 3-мя параметрами, например углами Эйлера
 - Применить вращение, выраженное через углы Эйлера – много тригонометрических функций, медленно

Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много
 - Применить матрицу к вектору – минимум операций сложения и умножения
 - Композиция вращений – произведение матриц, довольно быстрая операция
 - Интерполяция между двумя матрицами вращения – почти всегда **не** матрица вращения
- Вращения образуют 3х-мерную группу $SO(3)$, т.е. описываются 3-мя параметрами, например углами Эйлера
 - Применить вращение, выраженное через углы Эйлера – много тригонометрических функций, медленно
 - Композиция таких вращений – очень сложная операция, много тригонометрических функций

Анимирование вращений

- Обычно мы представляли вращения матрицами:
 - Матрица 3×3 – 9 значений, это много
 - Применить матрицу к вектору – минимум операций сложения и умножения
 - Композиция вращений – произведение матриц, довольно быстрая операция
 - Интерполяция между двумя матрицами вращения – почти всегда **не** матрица вращения
- Вращения образуют 3х-мерную группу $SO(3)$, т.е. описываются 3-мя параметрами, например углами Эйлера
 - Применить вращение, выраженное через углы Эйлера – много тригонометрических функций, медленно
 - Композиция таких вращений – очень сложная операция, много тригонометрических функций
- Хочется компромисс между сложностью, объёмом хранения и удобством использования

- Кватернионы \mathbb{H} – четырёхмерная *некоммутативная* алгебра над вещественными числами с тремя мнимыми единицами i, j, k
- Каждый элемент $q \in \mathbb{H}$ представляется в виде $q = a + bi + cj + dk$, где $a, b, c, d \in \mathbb{R}$ – коэффициенты кватерниона

- Кватернионы \mathbb{H} – четырёхмерная *некоммутативная* алгебра над вещественными числами с тремя мнимыми единицами i, j, k
- Каждый элемент $q \in \mathbb{H}$ представляется в виде $q = a + bi + cj + dk$, где $a, b, c, d \in \mathbb{R}$ – коэффициенты кватерниона
- Правила умножения:
 - $i^2 = j^2 = k^2 = -1$
 - $ij = -ji = k$
 - $jk = -kj = i$
 - $ki = -ik = j$

- Сопряжённый кватернион определяется как
$$\bar{q} = a - bi - cj - dk$$
- Норма кватерниона: число $q \cdot \bar{q} = a^2 + b^2 + c^2 + d^2 = |q|^2$
- Обратный кватернион: $q^{-1} = \frac{1}{|q|^2} \bar{q}$
- Единичный кватернион: $|q| = 1$
- Единичные кватернионы образуют трёхмерную сферу \mathbb{S}^3 (вложенную в четырёхмерное пространство $\mathbb{R}^4 \cong \mathbb{H}$)

Кватернионы: альтернативное представление

- Для кватерниона $q = a + bi + cj + dk$ назовём его скалярной частью число a , а векторной частью – вектор $v = (b, c, d)$
- Кватернион – пара скаляр + вектор: $q = (a, v)$

Кватернионы: альтернативное представление

- Для кватерниона $q = a + bi + cj + dk$ назовём его скалярной частью число a , а векторной частью – вектор $v = (b, c, d)$
- Кватернион – пара скаляр + вектор: $q = (a, v)$
- Сопряжённый кватернион: $(a, -v)$

Кватернионы: альтернативное представление

- Для кватерниона $q = a + bi + cj + dk$ назовём его скалярной частью число a , а векторной частью – вектор $v = (b, c, d)$
- Кватернион – пара скаляр + вектор: $q = (a, v)$
- Сопряжённый кватернион: $(a, -v)$
- Произведение кватернионов:
$$(a_1, v_1) \cdot (a_2, v_2) = (a_1 \cdot a_2 - v_1 \cdot v_2, a_1 v_2 + a_2 v_1 + v_1 \times v_2)$$

Кватернионы: альтернативное представление

- Для кватерниона $q = a + bi + cj + dk$ назовём его скалярной частью число a , а векторной частью – вектор $v = (b, c, d)$
- Кватернион – пара скаляр + вектор: $q = (a, v)$
- Сопряжённый кватернион: $(a, -v)$
- Произведение кватернионов:
$$(a_1, v_1) \cdot (a_2, v_2) = (a_1 \cdot a_2 - v_1 \cdot v_2, a_1 v_2 + a_2 v_1 + v_1 \times v_2)$$
- В таком виде кватернионы удобно реализовывать в шейдерах

- Представим произвольный трёхмерный вектор v как кватернион с нулевой скалярной частью $(0, v)$

- Представим произвольный трёхмерный вектор v как кватернион с нулевой скалярной частью $(0, v)$
- Вращение вокруг оси w (единичный вектор) на угол θ можно реализовать как $q \cdot (0, v) \cdot q^{-1}$, где $q = (\cos \frac{\theta}{2}, w \cdot \sin \frac{\theta}{2})$

- Представим произвольный трёхмерный вектор v как кватернион с нулевой скалярной частью $(0, v)$
- Вращение вокруг оси w (единичный вектор) на угол θ можно реализовать как $q \cdot (0, v) \cdot q^{-1}$, где $q = (\cos \frac{\theta}{2}, w \cdot \sin \frac{\theta}{2})$
- NB: $|q| = 1$
- NB: $q^{-1} = (\cos \frac{\theta}{2}, -w \cdot \sin \frac{\theta}{2})$

Кватернионы: вращения

- Представим произвольный трёхмерный вектор v как кватернион с нулевой скалярной частью $(0, v)$
- Вращение вокруг оси w (единичный вектор) на угол θ можно реализовать как $q \cdot (0, v) \cdot q^{-1}$, где $q = (\cos \frac{\theta}{2}, w \cdot \sin \frac{\theta}{2})$
- **NB:** $|q| = 1$
- **NB:** $q^{-1} = (\cos \frac{\theta}{2}, -w \cdot \sin \frac{\theta}{2})$
- Любое вращение представляется единичным кватернионом, и любой единичный кватернион описывает вращение

Кватернионы: вращения

- Представим произвольный трёхмерный вектор v как кватернион с нулевой скалярной частью $(0, v)$
- Вращение вокруг оси w (единичный вектор) на угол θ можно реализовать как $q \cdot (0, v) \cdot q^{-1}$, где $q = (\cos \frac{\theta}{2}, w \cdot \sin \frac{\theta}{2})$
- NB: $|q| = 1$
- NB: $q^{-1} = (\cos \frac{\theta}{2}, -w \cdot \sin \frac{\theta}{2})$
- Любое вращение представляется единичным кватернионом, и любой единичный кватернион описывает вращение
- q и $-q$ описывают одно и то же вращение (и только они)

- Для вращения нужны только алгебраические операции \implies быстро!

- Для вращения нужны только алгебраические операции \implies быстро!
- Композиция вращений – произведение кватернионов:
$$q_2 \cdot (q_1 \cdot (0, v) \cdot q_1^{-1}) \cdot q_2^{-1} = (q_2 \cdot q_1) \cdot (0, v) \cdot (q_1^{-1} \cdot q_2^{-1}) =$$
$$(q_2 q_1) \cdot (0, v) \cdot (q_2 q_1)^{-1}$$

Кватернионы: вращения

- Для вращения нужны только алгебраические операции \implies быстро!
- Композиция вращений – произведение кватернионов:
$$q_2 \cdot (q_1 \cdot (0, v) \cdot q_1^{-1}) \cdot q_2^{-1} = (q_2 \cdot q_1) \cdot (0, v) \cdot (q_1^{-1} \cdot q_2^{-1}) =$$
$$(q_2 q_1) \cdot (0, v) \cdot (q_2 q_1)^{-1}$$
- Стандартный способ представления вращений объектов в 3D движках

- Линейная интерполяция двух единичных кватернионов – почти всегда **не** единичный кватернион

Сферическая интерполяция

- Линейная интерполяция двух единичных кватернионов – почти всегда **не** единичный кватернион
- Можно отнормировать результат, но это не будет соответствовать равномерной интерполяции

Сферическая интерполяция

- Линейная интерполяция двух единичных кватернионов – почти всегда **не** единичный кватернион
- Можно отнормировать результат, но это не будет соответствовать равномерной интерполяции
- Правильный способ: использовать геодезическую кривую на поверхности сферы

Сферическая интерполяция

- Линейная интерполяция двух единичных кватернионов – почти всегда **не** единичный кватернион
- Можно отнормировать результат, но это не будет соответствовать равномерной интерполяции
- Правильный способ: использовать геодезическую кривую на поверхности сферы
- Эта операция называется `slerp` (*spherical linear interpolation*), имеет явную формулу и реализована в большинстве математических библиотек (в т.ч. `glm`)

- Есть два варианта представить кватернион $q = w + xi + yj + zk$:

Кватернионы: представление в коде

- Есть два варианта представить кватернион

$$q = w + xi + yj + zk:$$

- Как четырёхмерный вектор $[w, x, y, z]$ – логичнее с математической точки зрения
- Как четырёхмерный вектор $[x, y, z, w]$ – удобнее работать в GLSL (и других шейдерных языках)

Кватернионы: представление в коде

- Есть два варианта представить кватернион $q = w + xi + yj + zk$:
 - Как четырёхмерный вектор $[w, x, y, z]$ – логичнее с математической точки зрения
 - Как четырёхмерный вектор $[x, y, z, w]$ – удобнее работать в GLSL (и других шейдерных языках)
- Общепринятого варианта нет
- В библиотеке glm – $[w, x, y, z]$ (есть **наполовину сломанная** поддержка $[x, y, z, w]$)
- Формат моделей glTF описывает вращения как $[x, y, z, w]$

- en.wikipedia.org/wiki/Quaternion
- en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
- en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
- en.wikipedia.org/wiki/Slerp

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования
- Обычно это поворот + масштабирование + сдвиг

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования
- Обычно это поворот + масштабирование + сдвиг
- Поворот – кватернион q

Преобразования объектов

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования
- Обычно это поворот + масштабирование + сдвиг
- Поворот – кватернион q
- Масштабирование – одно число s (*изотропное масштабирование*) или три числа (разный масштаб по разным осям)

Преобразования объектов

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования
- Обычно это поворот + масштабирование + сдвиг
- Поворот – кватернион q
- Масштабирование – одно число s (*изотропное масштабирование*) или три числа (разный масштаб по разным осям)
- Сдвиг – вектор сдвига t

Преобразования объектов

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования
- Обычно это поворот + масштабирование + сдвиг
- Поворот – кватернион q
- Масштабирование – одно число s (*изотропное масштабирование*) или три числа (разный масштаб по разным осям)
- Сдвиг – вектор сдвига t
- Преобразование вершин: $v \mapsto s \cdot qvq^{-1} + t$

Преобразования объектов

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования
- Обычно это поворот + масштабирование + сдвиг
- Поворот – кватернион q
- Масштабирование – одно число s (*изотропное масштабирование*) или три числа (разный масштаб по разным осям)
- Сдвиг – вектор сдвига t
- Преобразование вершин: $v \mapsto s \cdot qvq^{-1} + t$
- Преобразование нормалей: $n \mapsto qnq^{-1}$ либо $n \mapsto \frac{s \cdot qnq^{-1}}{\|s \cdot qnq^{-1}\|}$

- Часто объекты сцены/мира образуют иерархическую структуру (шапка на человеке, человек в машине, машина на корабле)

- Часто объекты сцены/мира образуют иерархическую структуру (шапка на человеке, человек в машине, машина на корабле)
- Удобно описывать полное преобразование объекта (позиция + поворот + масштабирование) не относительно центра сцены/мира, а относительно родительского объекта

- Часто объекты сцены/мира образуют иерархическую структуру (шапка на человеке, человек в машине, машина на корабле)
- Удобно описывать полное преобразование объекта (позиция + поворот + масштабирование) не относительно центра сцены/мира, а относительно родительского объекта
- Обычно в такой ситуации есть один корневой объект – сцена

- Часто объекты сцены/мира образуют иерархическую структуру (шапка на человеке, человек в машине, машина на корабле)
- Удобно описывать полное преобразование объекта (позиция + поворот + масштабирование) не относительно центра сцены/мира, а относительно родительского объекта
- Обычно в такой ситуации есть один корневой объект – сцена
- \implies Нужно уметь вычислять итоговое преобразование объекта, т.е. композицию всех преобразований от корня до нашего объекта

Композиция аффинных преобразований

$$(t_2, s_2, q_2) \cdot (t_1, s_1, q_1) \cdot v = (t, s, q) \cdot v \quad (1)$$

$$\begin{aligned}(t_2, s_2, q_2) \cdot (t_1, s_1, q_1) \cdot v &= s_2 q_2 (s_1 q_1 v q_1^{-1} + t_1) q_2^{-1} + t_2 = \\ &= s_2 s_1 (q_2 q_1) v (q_2 q_1)^{-1} + s_2 q_2 t_1 q_2^{-1} + t_2 = \\ &= (s_2 q_2 t_1 q_2^{-1} + t_2, s_2 s_1, q_2 q_1) \cdot v\end{aligned} \quad (2)$$

$$q = q_2 q_1 \quad (3)$$

$$s = s_2 s_1 \quad (4)$$

$$t = s_2 q_2 t_1 q_2^{-1} + t_2 \quad (5)$$

- Анимация положения объекта в пространстве – неплохо, но скучно

- Анимация положения объекта в пространстве – неплохо, но скучно
- Хочется анимировать саму модель, т.е. двигать её вершины

- Анимация положения объекта в пространстве – неплохо, но скучно
- Хочется анимировать саму модель, т.е. двигать её вершины
- 2 способа:
 - Покадровая анимация (*morph-target animation*)
 - Скелетная анимация (*skeletal animation*)

- Анимируются в явном виде все вершины по отдельности

Покадровая анимация моделей

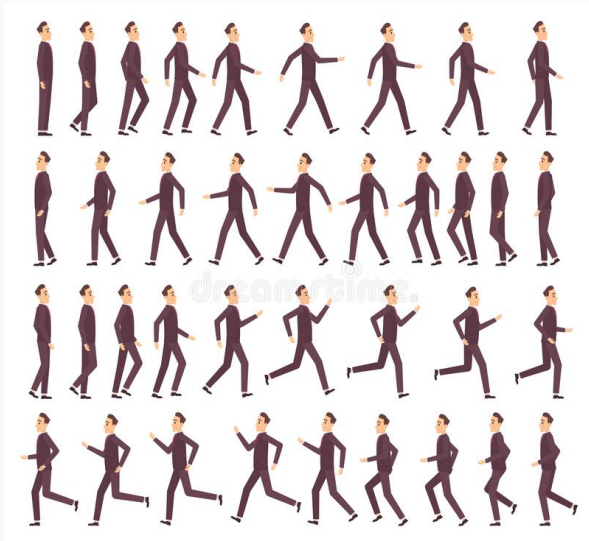
- Анимируются в явном виде все вершины по отдельности
- Анимация хранится в виде 'кадров': фиксированных состояний модели (наборов координат вершин)

Покадровая анимация моделей

- Анимируются в явном виде все вершины по отдельности
- Анимация хранится в виде 'кадров': фиксированных состояний модели (наборов координат вершин)
- Вершинный шейдер принимает два набора атрибутов позиций вершин и интерполирует между ними
 - Интерполяция может использовать easing functions

- Анимируются в явном виде все вершины по отдельности
- Анимация хранится в виде 'кадров': фиксированных состояний модели (наборов координат вершин)
- Вершинный шейдер принимает два набора атрибутов позиций вершин и интерполирует между ними
 - Интерполяция может использовать easing functions
- Фактически, это сплайн, значение которого – набор координат всех вершин

Покадровая анимация моделей



Покадровая анимация моделей

- Много вариантов реализации:
 - При смене кадра анимации загружать в VBO новые данные
 - При смене кадра менять VBO/VAO
 - Хранить кадры отдельно (например, в buffer textures), передавать в шейдер только номер кадра

Покадровая анимация моделей

- Много вариантов реализации:
 - При смене кадра анимации загружать в VBO новые данные
 - При смене кадра менять VBO/VAO
 - Хранить кадры отдельно (например, в buffer textures), передавать в шейдер только номер кадра
- Много проблем:

Покадровая анимация моделей

- Много вариантов реализации:
 - При смене кадра анимации загружать в VBO новые данные
 - При смене кадра менять VBO/VAO
 - Хранить кадры отдельно (например, в buffer textures), передавать в шейдер только номер кадра
- Много проблем:
 - — Сложно модифицировать: нужно двигать все вершины модели

Покадровая анимация моделей

- Много вариантов реализации:
 - При смене кадра анимации загружать в VBO новые данные
 - При смене кадра менять VBO/VAO
 - Хранить кадры отдельно (например, в buffer textures), передавать в шейдер только номер кадра
- Много проблем:
 - — Сложно модифицировать: нужно двигать все вершины модели
 - — Сложно добавлять процедурную анимацию (напр. поворачивать голову в нужную сторону)

Покадровая анимация моделей

- Много вариантов реализации:
 - При смене кадра анимации загружать в VBO новые данные
 - При смене кадра менять VBO/VAO
 - Хранить кадры отдельно (например, в buffer textures), передавать в шейдер только номер кадра
- Много проблем:
 - — Сложно модифицировать: нужно двигать все вершины модели
 - — Сложно добавлять процедурную анимацию (напр. поворачивать голову в нужную сторону)
 - — Требуется много памяти

Покадровая анимация моделей

- Много вариантов реализации:
 - При смене кадра анимации загружать в VBO новые данные
 - При смене кадра менять VBO/VAO
 - Хранить кадры отдельно (например, в buffer textures), передавать в шейдер только номер кадра
- Много проблем:
 - — Сложно модифицировать: нужно двигать все вершины модели
 - — Сложно добавлять процедурную анимацию (напр. поворачивать голову в нужную сторону)
 - — Требуется много памяти
 - — Для хорошего качества нужно много кадров, иначе будут артефакты интерполяции (напр. модель начнёт пересекать саму себя)

Покадровая анимация моделей

- Много вариантов реализации:
 - При смене кадра анимации загружать в VBO новые данные
 - При смене кадра менять VBO/VAO
 - Хранить кадры отдельно (например, в buffer textures), передавать в шейдер только номер кадра
- Много проблем:
 - — Сложно модифицировать: нужно двигать все вершины модели
 - — Сложно добавлять процедурную анимацию (напр. поворачивать голову в нужную сторону)
 - — Требуется много памяти
 - — Для хорошего качества нужно много кадров, иначе будут артефакты интерполяции (напр. модель начнёт пересекать саму себя)
- Обычно **не** используется для 3D моделей

Скелетная анимация моделей

- Модель привязывается к виртуальному *скелету*

Скелетная анимация моделей

- Модель привязывается к виртуальному *скелету*
- Скелет – иерархия виртуальных *костей*

Скелетная анимация моделей

- Модель привязывается к виртуальному *скелету*
- Скелет – иерархия виртуальных *костей*
- Каждая вершина привязана к одной или (чаще) нескольким костям

Скелетная анимация моделей

- Модель привязывается к виртуальному *скелету*
- Скелет – иерархия виртуальных *костей*
- Каждая вершина привязана к одной или (чаще) нескольким костям
- Каждой паре вершина-кость соответствует некоторый вес: насколько эта кость влияет на эту вершину (сумма весов для одной вершины должна равняться 1)

Скелетная анимация моделей

- Модель привязывается к виртуальному *скелету*
- Скелет – иерархия виртуальных *костей*
- Каждая вершина привязана к одной или (чаще) нескольким костям
- Каждой паре вершина-кость соответствует некоторый вес: насколько эта кость влияет на эту вершину (сумма весов для одной вершины должна равняться 1)
- Кадры анимации задаются только для скелета (исходная модель существует в одном экземпляре)

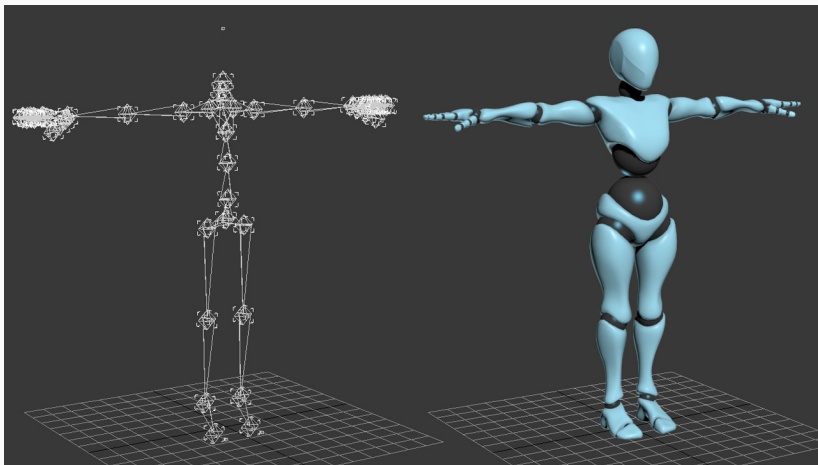
Скелетная анимация моделей

- Модель привязывается к виртуальному скелету
- Скелет – иерархия виртуальных костей
- Каждая вершина привязана к одной или (чаще) нескольким костям
- Каждой паре вершина-кость соответствует некоторый вес: насколько эта кость влияет на эту вершину (сумма весов для одной вершины должна равняться 1)
- Кадры анимации задаются только для скелета (исходная модель существует в одном экземпляре)
- Интерполируются только преобразования костей (костей гораздо меньше, чем вершин \implies это не страшно делать даже на CPU)

Скелетная анимация моделей

- Модель привязывается к виртуальному скелету
- Скелет – иерархия виртуальных костей
- Каждая вершина привязана к одной или (чаще) нескольким костям
- Каждой паре вершина-кость соответствует некоторый вес: насколько эта кость влияет на эту вершину (сумма весов для одной вершины должна равняться 1)
- Кадры анимации задаются только для скелета (исходная модель существует в одном экземпляре)
- Интерполируются только преобразования костей (костей гораздо меньше, чем вершин \implies это не страшно делать даже на CPU)
- В вершинном шейдере вычисляется итоговое преобразование для вершины как среднее между преобразованиями связанных с ней костей

Скелетная анимация моделей



Скелетная анимация моделей

```
uniform mat4x4 bones[16];

layout (location = 0) in vec4 in_position;
layout (location = 1) in ivec2 in_joints;
layout (location = 2) in vec2 in_weights;

void main()
{
    gl_Position =
        in_weights.x * bones[in_joints.x] * in_position
        + in_weights.y * bones[in_joints.y] * in_position;
}
```

- К нормальям тоже нужно применять преобразования (но не сдвиги!)

Скелетная анимация моделей

- К нормальям тоже нужно применять преобразования (но не сдвиги!)
- Кости обычно тоже выстроены в иерархию \implies перед применением нужно вычислить суммарное преобразование (композицию)

Скелетная анимация моделей

- К нормальям тоже нужно применять преобразования (но не сдвиги!)
- Кости обычно тоже выстроены в иерархию \implies перед применением нужно вычислить суммарное преобразование (композицию)
- Преобразования костей часто заданы в локальной для кости системе координат для удобства \implies перед применением нужно умножить на преобразование, переводящее из системы координат модели в локальную систему координат кости (*inverse bind matrix* в формате glTF)

Скелетная анимация моделей

- К нормальям тоже нужно применять преобразования (но не сдвиги!)
- Кости обычно тоже выстроены в иерархию \implies перед применением нужно вычислить суммарное преобразование (композицию)
- Преобразования костей часто заданы в локальной для кости системе координат для удобства \implies перед применением нужно умножить на преобразование, переводящее из системы координат модели в локальную систему координат кости (*inverse bind matrix* в формате `glTF`)
- Нужно быть внимательным к особенностям задания преобразований и системам координат в разных редакторах и форматах!

- Обычно вычисление преобразования для кости выглядит как

```
...  
* bone.parent.parent.local_transform  
* bone.parent.local_transform *  
* bone.local_transform  
* bone.inverse_bind_matrix
```

- + Удобно и интуитивно модифицировать (все 3D-редакторы имеют поддержку скелетных анимаций)

- + Удобно и интуитивно модифицировать (все 3D-редакторы имеют поддержку скелетных анимаций)
- + Небольшой расход памяти (модель не дублируется)

- + Удобно и интуитивно модифицировать (все 3D-редакторы имеют поддержку скелетных анимаций)
- + Небольшой расход памяти (модель не дублируется)
- + Легко добавлять процедурную анимацию

- + Удобно и интуитивно модифицировать (все 3D-редакторы имеют поддержку скелетных анимаций)
- + Небольшой расход памяти (модель не дублируется)
- + Легко добавлять процедурную анимацию
- Самый распространённый способ анимировать модели

- Откуда брать скелетную анимацию?

- Откуда брать скелетную анимацию?
 - Сплайны – так обычно описываются анимации в 3D редакторах и форматах моделей (отдельный сплайн для вращения, масштаба и сдвига каждой кости)

- Откуда брать скелетную анимацию?
 - Сплайны – так обычно описываются анимации в 3D редакторах и форматах моделей (отдельный сплайн для вращения, масштаба и сдвига каждой кости)
 - Процедурная анимация – код, генерирующий анимацию на лету

- Вычисление координат вершины по известным преобразованиям костей называется *forward kinematics*

- Вычисление координат вершины по известным преобразованиям костей называется *forward kinematics*
- Часто хочется уметь решать обратную задачу: по финальному положению вершины вычислить подходящие преобразования костей

- Вычисление координат вершины по известным преобразованиям костей называется *forward kinematics*
- Часто хочется уметь решать обратную задачу: по финальному положению вершины вычислить подходящие преобразования костей
 - Повернуть голову, чтобы она смотрела в нужном направлении

- Вычисление координат вершины по известным преобразованиям костей называется *forward kinematics*
- Часто хочется уметь решать обратную задачу: по финальному положению вершины вычислить подходящие преобразования костей
 - Повернуть голову, чтобы она смотрела в нужном направлении
 - Поставить ногу на ландшафт с неизвестным заранее наклоном

- Вычисление координат вершины по известным преобразованиям костей называется *forward kinematics*
- Часто хочется уметь решать обратную задачу: по финальному положению вершины вычислить подходящие преобразования костей
 - Повернуть голову, чтобы она смотрела в нужном направлении
 - Поставить ногу на ландшафт с неизвестным заранее наклоном
 - Повернуть руку так, чтобы кисть схватила нужный объект

- Вычисление координат вершины по известным преобразованиям костей называется *forward kinematics*
- Часто хочется уметь решать обратную задачу: по финальному положению вершины вычислить подходящие преобразования костей
 - Повернуть голову, чтобы она смотрела в нужном направлении
 - Поставить ногу на ландшафт с неизвестным заранее наклоном
 - Повернуть руку так, чтобы кисть схватила нужный объект
- Удобно при формировании анимации в редакторе; необходимо для нетривиальных процедурных анимаций

- Вычисление координат вершины по известным преобразованиям костей называется *forward kinematics*
- Часто хочется уметь решать обратную задачу: по финальному положению вершины вычислить подходящие преобразования костей
 - Повернуть голову, чтобы она смотрела в нужном направлении
 - Поставить ногу на ландшафт с неизвестным заранее наклоном
 - Повернуть руку так, чтобы кисть схватила нужный объект
- Удобно при формировании анимации в редакторе; необходимо для нетривиальных процедурных анимаций
- Эта (обратная) задача называется *inverse kinematics* (IK)

- Финальное положение точки – функция от преобразований отдельных костей

- Финальное положение точки – функция от преобразований отдельных костей
- Часто кости могут только вращаться (напр. тело человека), но не сдвигаться или масштабироваться

- Финальное положение точки – функция от преобразований отдельных костей
- Часто кости могут только вращаться (напр. тело человека), но не сдвигаться или масштабироваться
- Тогда положение точки – функция углов $p = f(\theta_1, \theta_2, \dots, \theta_n)$

- Финальное положение точки – функция от преобразований отдельных костей
- Часто кости могут только вращаться (напр. тело человека), но не сдвигаться или масштабироваться
- Тогда положение точки – функция углов $p = f(\theta_1, \theta_2, \dots, \theta_n)$
- p известна, $\{\theta_i\}$ неизвестны \implies ИК сводится к задаче решения нелинейной системы уравнений

- Финальное положение точки – функция от преобразований отдельных костей
- Часто кости могут только вращаться (напр. тело человека), но не сдвигаться или масштабироваться
- Тогда положение точки – функция углов $p = f(\theta_1, \theta_2, \dots, \theta_n)$
- p известна, $\{\theta_i\}$ неизвестны \implies ИК сводится к задаче решения нелинейной системы уравнений
- В некоторых простых частных случаях можно решить явно (напр. поровот головы – atan2 от вектора направления)

- Финальное положение точки – функция от преобразований отдельных костей
- Часто кости могут только вращаться (напр. тело человека), но не сдвигаться или масштабироваться
- Тогда положение точки – функция углов $p = f(\theta_1, \theta_2, \dots, \theta_n)$
- p известна, $\{\theta_i\}$ неизвестны \implies ИК сводится к задаче решения нелинейной системы уравнений
- В некоторых простых частных случаях можно решить явно (напр. поровот головы – atan2 от вектора направления)
- В общем случае решается итеративными методами (напр. многомерным методом Ньютона)

- Если для модели задано несколько анимаций (ходьба, бег, прыжок, поворот, и т.д.) нужно уметь между ними переключаться

Скелетная анимация

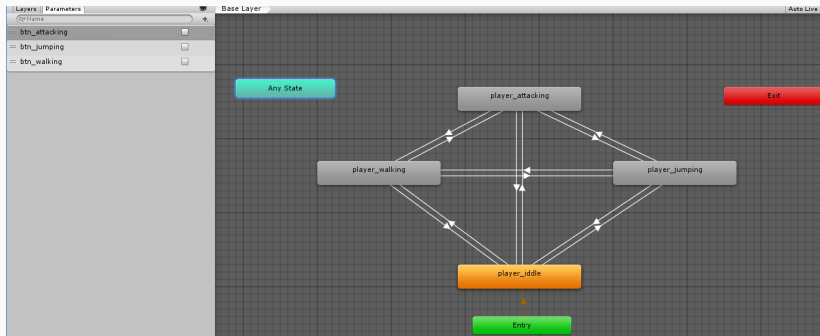
- Если для модели задано несколько анимаций (ходьба, бег, прыжок, поворот, и т.д.) нужно уметь между ними переключаться
- Обычно достаточно интерполировать `local_transform` каждой кости от первой ко второй анимации

Скелетная анимация

- Если для модели задано несколько анимаций (ходьба, бег, прыжок, поворот, и т.д.) нужно уметь между ними переключаться
- Обычно достаточно интерполировать `local_transform` каждой кости от первой ко второй анимации
- Не все переходы имеют смысл (напр. люди обычно не переходят в лежачее положение сразу из бега или прыжка)
⇒ часто используют state-машины анимаций с настраиваемыми переходами между состояниями

- Если для модели задано несколько анимаций (ходьба, бег, прыжок, поворот, и т.д.) нужно уметь между ними переключаться
- Обычно достаточно интерполировать `local_transform` каждой кости от первой ко второй анимации
- Не все переходы имеют смысл (напр. люди обычно не переходят в лежачее положение сразу из бега или прыжка)
⇒ часто используют state-машины анимаций с настраиваемыми переходами между состояниями
- В современности почти всегда используется комбинированный подход: state-машина для переходов между заранее подготовленными анимациями + специфичная для ситуации процедурная анимация

State-машина для анимации



- Есть способы для некоторых ситуаций полностью процедурно генерировать анимацию

- Есть способы для некоторых ситуаций полностью процедурно генерировать анимацию
- Анимация передвижения пауков:
`youtube.com/watch?v=e6Gjhr1IP6w`

- Есть способы для некоторых ситуаций полностью процедурно генерировать анимацию
- Анимация передвижения пауков:
youtube.com/watch?v=e6Gjhr1IP6w
- Оффлайн генерация анимации движения для двуногих:
youtube.com/watch?v=pgaEE27nsQw

- en.wikipedia.org/wiki/Skeletal_animation
- en.wikipedia.org/wiki/Inverse_kinematics
- learnopengl.com/Guest-Articles/2020/Skeletal-Animation
- ogldev.org/www/tutorial38/tutorial38.html
- Skeletal animation in glTF
- Видео-тutorиал по скелетной анимации
- Видео про графы анимаций в крупных движках

- Существует колоссально много форматов для 3D-моделей

- Существует колоссально много форматов для 3D-моделей
- Они отличаются

- Существует колоссально много форматов для 3D-моделей
- Они отличаются
 - Набором атрибутов вершин (вторые текстурные координаты, веса для анимации)

- Существует колоссально много форматов для 3D-моделей
- Они отличаются
 - Набором атрибутов вершин (вторые текстурные координаты, веса для анимации)
 - Поддержкой материалов и их сложностью

- Существует колоссально много форматов для 3D-моделей
- Они отличаются
 - Набором атрибутов вершин (вторые текстурные координаты, веса для анимации)
 - Поддержкой материалов и их сложностью
 - Поддержкой иерархии объектов и сцен

- Существует колоссально много форматов для 3D-моделей
- Они отличаются
 - Набором атрибутов вершин (вторые текстурные координаты, веса для анимации)
 - Поддержкой материалов и их сложностью
 - Поддержкой иерархии объектов и сцен
 - Поддержкой анимаций

- .x – устаревший формат, часто использовавшийся вместе с DirectX

Форматы 3D моделей

- .x – устаревший формат, часто использовавшийся вместе с DirectX
- .3ds – устаревший формат, использовавшийся в девяностых и нулевых, не поддерживал анимацию и нормали

Форматы 3D моделей

- `.x` – устаревший формат, часто использовавшийся вместе с DirectX
- `.3ds` – устаревший формат, использовавшийся в девяностых и нулевых, не поддерживал анимацию и нормали
- `.obj` – текстовый формат, использующийся из-за своей простоты, не поддерживает нестандартных атрибутов и анимацию

Форматы 3D моделей

- .x – устаревший формат, часто использовавшийся вместе с DirectX
- .3ds – устаревший формат, использовавшийся в девяностых и нулевых, не поддерживал анимацию и нормали
- .obj – текстовый формат, использующийся из-за своей простоты, не поддерживает нестандартных атрибутов и анимацию
- .dae (COLLADA) – распространённый современный XML-формат, поддерживает почти всё

Форматы 3D моделей

- .x – устаревший формат, часто использовавшийся вместе с DirectX
- .3ds – устаревший формат, использовавшийся в девяностых и нулевых, не поддерживал анимацию и нормали
- .obj – текстовый формат, использующийся из-за своей простоты, не поддерживает нестандартных атрибутов и анимацию
- .dae (COLLADA) – распространённый современный XML-формат, поддерживает почти всё
 - Из-за многословности XML формат занимает очень много памяти; обычно используется как промежуточный перед конвертацией в специфичный для движка бинарный формат

Форматы 3D моделей

- `.x` – устаревший формат, часто использовавшийся вместе с DirectX
- `.3ds` – устаревший формат, использовавшийся в девяностых и нулевых, не поддерживал анимацию и нормали
- `.obj` – текстовый формат, использующийся из-за своей простоты, не поддерживает нестандартных атрибутов и анимацию
- `.dae` (COLLADA) – распространённый современный XML-формат, поддерживает почти всё
 - Из-за многословности XML формат занимает очень много памяти; обычно используется как промежуточный перед конвертацией в специфичный для движка бинарный формат
- `.gltf` – современный JSON-формат, разработанный Khronos Group; поддерживает нетривиальные атрибуты, материалы и анимацию; в JSON хранится логическое описание данных, а сами бинарные данные (вершины, индексы, анимации) могут храниться в отдельных файлах; очень удобен для загрузки и использования в OpenGL

- Документация по COLLADA
- Документация по glTF
- Список 3D форматов