Компьютерная графика

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

2025

Анимация

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

Анимация

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

Анимация

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

Анимация

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

Анимация

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

Анимация

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

Анимация

- 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

Анимация

- Зачем нужна анимация?

Анимация

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

Анимация

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

Анимация: общие идеи

- 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 = lerp(old_x, new_x, 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 = lerp(old_x, new_x, 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 = lerp(old_x, new_x, time)`
    - \+ Легко настраивать форму интерполяции (*easing functions*)
    - — Нужно обрабатывать ситуацию, когда новое значение изменилось в процессе анимации

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 2**: обновлять `x` *только* на основе нового значения: `x = 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 = lerp(x, new_x, speed * dt)`
    - Если `dt > 1/speed`, возникнет нестабильность
    - Более точная, стабильная формула: `x = lerp(x, new_x, 1 - 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 = lerp(x, new_x, speed * dt)`
  - Если `dt > 1/speed`, возникнет нестабильность
  - Более точная, стабильная формула:
    `x = lerp(x, new_x, 1 - exp(- speed * dt))`
  - Для точного вычисления `exp(x)-1` для маленьких значений `x` есть функция `std::expm1`

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 = lerp(x, new_x, speed * dt)`
  - Если `dt > 1/speed`, возникнет нестабильность
  - Более точная, стабильная формула:
    `x = lerp(x, new_x, 1 - exp(- speed * dt))`
  - Для точного вычисления `exp(x)-1` для маленьких значений `x` есть функция `std::expm1`
  - Решение дифференциального 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 = lerp(x, new_x, speed * dt)`
  - Если `dt > 1/speed`, возникнет нестабильность
  - Более точная, стабильная формула:
    `x = lerp(x, new_x, 1 - exp(- speed * dt))`
  - Для точного вычисления `exp(x)-1` для маленьких значений `x` есть функция `std::expm1`
  - Решение дифференциального 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 = lerp(x, new_x, speed * dt)`
  - Если `dt > 1/speed`, возникнет нестабильность
  - Более точная, стабильная формула:
    `x = lerp(x, new_x, 1 - exp(- speed * dt))`
  - Для точного вычисления `exp(x)-1` для маленьких значений `x` есть функция `std::expm1`
  - Решение дифференциального уравнения $\frac{dx}{dt} = speed \cdot (x_{new} - x)$
  - Удобно для анимации камеры, элементов интерфейса
  - Подробная статья про этот метод

Анимации: easing functions

- При временно́й интерполяции между двумя значениями вместо линейной интерполяции `x = lerp(old_x, new_x, t)` можно использовать т.н. easing functions

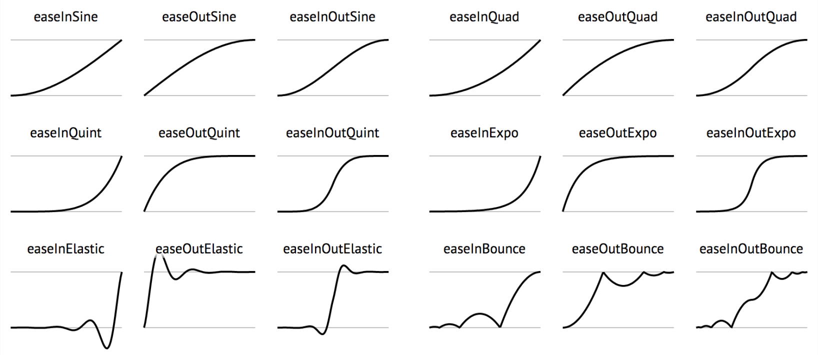
Анимации: easing functions

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

Анимации: easing functions

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

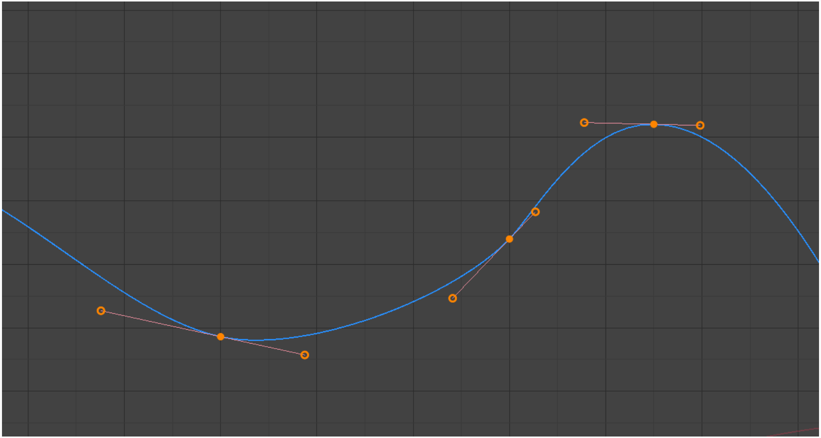
Анимация: сплайны

- 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 <u>не ограничивается</u> анимациями!
  - Curve fitting
  - Представление сложных геометрических форм (напр. зданий, шрифтов)
  - etc.

Анимация: сплайны



Bitmap-анимации

- Меняющееся со временем изображение

Bitmap-анимации

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

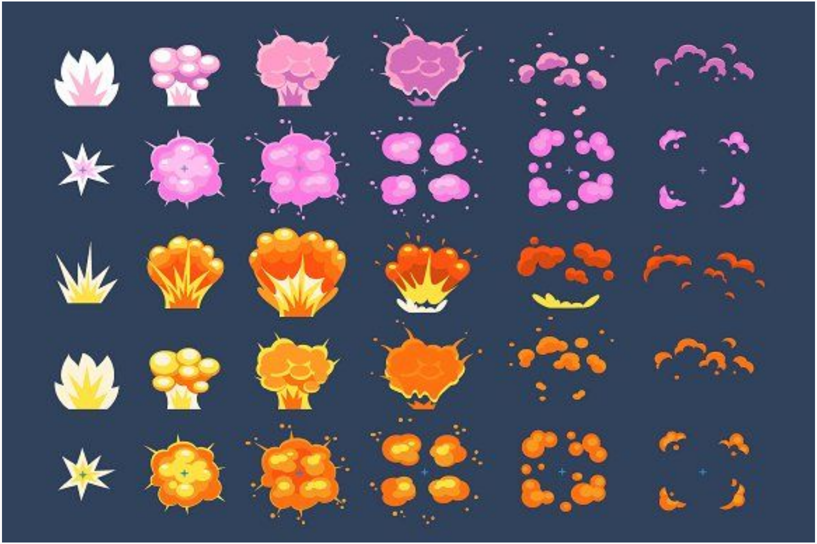
Bitmap-анимации

- 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

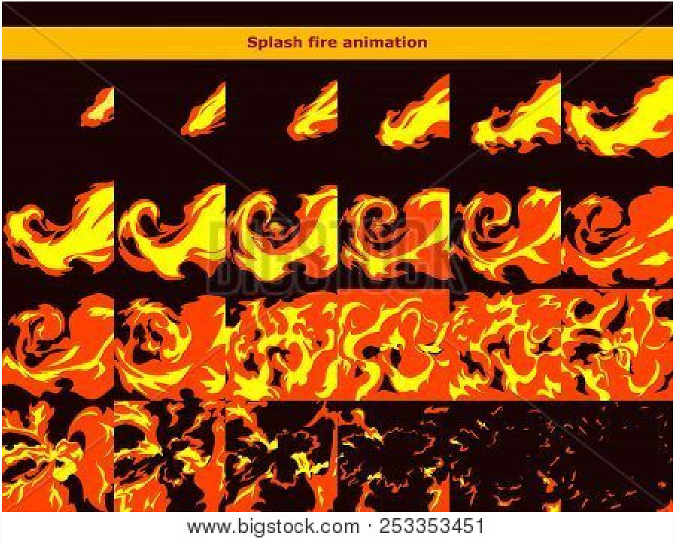
Bitmap-анимации

- 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



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 $\mathbb{SO}(3)$, 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 $\mathbb{SO}(3)$, 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 $\mathbb{SO}(3)$, 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 $\mathbb{SO}(3)$, 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

- 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

- 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:
  - $i^2 = j^2 = k^2 = -1$
  - $ij = -ji = k$
  - $jk = -kj = i$
  - $ki = -ik = j$

Кватернионы

- Сопряжённый кватернион определяется как $\overline{q} = a - bi - cj - dk$
- Норма кватерниона: число $q \cdot \overline{q} = a^2 + b^2 + c^2 + d^2 = |q|^2$
- Обратный кватернион: $q^{-1} = \frac{1}{|q|^2}\overline{q}$
- 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

Кватернионы: вращения

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

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 $q = (\cos\frac{\theta}{2}, w \cdot \sin\frac{\theta}{2})$

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: $q^{-1} = (\cos\frac{\theta}{2}, -w \cdot \sin\frac{\theta}{2})$

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)

Кватернионы: вращения

- Для вращения нужны только алгебраические операции $\implies$ быстро!

Кватернионы: вращения

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

Кватернионы: вращения

- Для вращения нужны только алгебраические операции $\Longrightarrow$ 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

Сферическая интерполяция

- 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`)

Кватернионы: представление в коде

- 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]`

Кватернионы: ссылки

- 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

Преобразования объектов

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

Преобразования объектов

- Часто для задания преобразования, применяемого к объекту, нам не нужна целиком матрица аффинного преобразования
- Обычно это масштабирование + поворот + сдвиг (в таком порядке!)

Преобразования объектов

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

Преобразования объектов

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

Преобразования объектов

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

Преобразования объектов

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

Преобразование нормалей

- В общем случае, если вершины модели преобразуются какой-то матрицей M, нормали могут **перестать** быть перпендикулярны к поверхности

Преобразование нормалей

- В общем случае, если вершины модели преобразуются какой-то матрицей M, нормали могут **перестать** быть перпендикулярны к поверхности
- Пусть n – нормаль к точке некой гладкой поверхности, и v – любой касательный вектор к поверхности в этой точке

Преобразование нормалей

- В общем случае, если вершины модели преобразуются какой-то матрицей M, нормали могут **перестать** быть перпендикулярны к поверхности
- Пусть n – нормаль к точке некой гладкой поверхности, и v – любой касательный вектор к поверхности в этой точке
- Из ортогональности $n \cdot v = 0$ не следует, что $(Mn) \cdot (Mv) = 0$!

Преобразование нормалей

- В общем случае, если вершины модели преобразуются какой-то матрицей M, нормали могут **перестать** быть перпендикулярны к поверхности
- Пусть n – нормаль к точке некой гладкой поверхности, и v – любой касательный вектор к поверхности в этой точке
- Из ортогональности $n \cdot v = 0$ не следует, что $(Mn) \cdot (Mv) = 0$!
- Но, по свойствам матриц и скалярного произведения $0 = n \cdot v = n \cdot (M^{-1}M)v = n \cdot M^{-1}(Mv) = (M^{-1})^T n \cdot (Mv)$

Преобразование нормалей

- В общем случае, если вершины модели преобразуются какой-то матрицей M, нормали могут **перестать** быть перпендикулярны к поверхности
- Пусть n – нормаль к точке некой гладкой поверхности, и v – любой касательный вектор к поверхности в этой точке
- Из ортогональности $n \cdot v = 0$ не следует, что $(Mn) \cdot (Mv) = 0$!
- Но, по свойствам матриц и скалярного произведения $0 = n \cdot v = n \cdot (M^{-1}M)v = n \cdot M^{-1}(Mv) = (M^{-1})^T n \cdot (Mv)$
- Таким образом, вектор $M^{-T}n$ будет перпендикулярен преобразованной поверхности

Преобразование нормалей

- В общем случае, если вершины модели преобразуются какой-то матрицей M, нормали могут **перестать** быть перпендикулярны к поверхности
- Пусть n – нормаль к точке некой гладкой поверхности, и v – любой касательный вектор к поверхности в этой точке
- Из ортогональности $n \cdot v = 0$ не следует, что $(Mn) \cdot (Mv) = 0$!
- Но, по свойствам матриц и скалярного произведения $0 = n \cdot v = n \cdot (M^{-1}M)v = n \cdot M^{-1}(Mv) = (M^{-1})^T n \cdot (Mv)$
- Таким образом, вектор $M^{-T}n$ будет перпендикулярен преобразованной поверхности
- $\Longrightarrow$ В общем случае нормали нужно преобразовывать матрицей M^{-T}

Преобразование нормалей

- Формула M^{-T} требует обращения матрицы (дорого) и нестабильна для преобразований с маленьким определителем (например, масштаб по одной из осей близок к нулю)

Преобразование нормалей

- Формула M^{-T} требует обращения матрицы (дорого) и нестабильна для преобразований с маленьким определителем (например, масштаб по одной из осей близок к нулю)
- Обратную матрицу A^{-1} можно посчитать по формуле $\frac{1}{\det A}\operatorname{adj} A$, где $\operatorname{adj} A$ – *присоединённая матрица*

Преобразование нормалей

- Формула M^{-T} требует обращения матрицы (дорого) и нестабильна для преобразований с маленьким определителем (например, масштаб по одной из осей близок к нулю)
- Обратную матрицу A^{-1} можно посчитать по формуле $\frac{1}{\det A} \operatorname{adj} A$, где $\operatorname{adj} A$ – *присоединённая матрица*
- $\operatorname{adj} A = (\operatorname{cof} A)^T$, где $\operatorname{cof} A$ – матрица *алгебраических дополнений*

Преобразование нормалей

- Формула M^{-T} требует обращения матрицы (дорого) и нестабильна для преобразований с маленьким определителем (например, масштаб по одной из осей близок к нулю)
- Обратную матрицу A^{-1} можно посчитать по формуле $\frac{1}{\det A}\operatorname{adj} A$, где $\operatorname{adj} A$ – *присоединённая матрица*
- $\operatorname{adj} A = (\operatorname{cof} A)^T$, где $\operatorname{cof} A$ – матрица *алгебраических дополнений*
- Подставляем $A = M^T$:

$$M^{-T} = \frac{1}{\det M^T}\operatorname{adj}(M^T) = \frac{1}{\det M}\operatorname{cof} M$$

Преобразование нормалей

- Формула M^{-T} требует обращения матрицы (дорого) и нестабильна для преобразований с маленьким определителем (например, масштаб по одной из осей близок к нулю)
- Обратную матрицу A^{-1} можно посчитать по формуле $\frac{1}{\det A} \operatorname{adj} A$, где $\operatorname{adj} A$ – *присоединённая матрица*
- $\operatorname{adj} A = (\operatorname{cof} A)^T$, где $\operatorname{cof} A$ – матрица *алгебраических дополнений*
- Подставляем $A = M^T$:

$$M^{-T} = \frac{1}{\det M^T} \operatorname{adj}(M^T) = \frac{1}{\det M} \operatorname{cof} M$$

- Определитель $\det M$ можно игнорировать: это просто число, оно исчезнет, когда мы отнормируем получившийся вектор нормали

Преобразование нормалей

- Формула M^{-T} требует обращения матрицы (дорого) и нестабильна для преобразований с маленьким определителем (например, масштаб по одной из осей близок к нулю)
- Обратную матрицу A^{-1} можно посчитать по формуле $\frac{1}{\det A}\operatorname{adj} A$, где $\operatorname{adj} A$ – *присоединённая матрица*
- $\operatorname{adj} A = (\operatorname{cof} A)^T$, где $\operatorname{cof} A$ – матрица *алгебраических дополнений*
- Подставляем $A = M^T$:

$$M^{-T} = \frac{1}{\det M^T}\operatorname{adj}(M^T) = \frac{1}{\det M}\operatorname{cof} M$$

- Определитель $\det M$ можно игнорировать: это просто число, оно исчезнет, когда мы отнормируем получившийся вектор нормали
- Остаётся только $\operatorname{cof} M$: она требует меньше вычислений, чем M^{-T}, и не чувствительна к вырожденным преобразованиям

Преобразование нормалей: частные случаи

- Если M – матрица вращения, то $M^T = M^{-1}$ и $M^{-T} = M$, т.е. нормали вращаются так же, как сами вершины

Преобразование нормалей: частные случаи

- Если M – матрица вращения, то $M^T = M^{-1}$ и $M^{-T} = M$, т.е. нормали вращаются так же, как сами вершины
- Если M – диагональная матрица неизотропного масштабирования, то $M^T = M$ и $M^{-T} = M^{-1}$, т.е. к нормалям применяется масштаб, обратный масштабу вершин

Иерархия объектов

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

Иерархия объектов

- 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

Иерархия объектов

- 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

Иерархия объектов

- 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
- $\Longrightarrow$ Нужно уметь вычислять итоговое преобразование объекта, т.е. композицию всех преобразований от корня до нашего объекта

Композиция аффинных преобразований

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

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

$$q = q_2 q_1 \tag{3}$$

$$s = s_2 s_1 \tag{4}$$

$$t = s_2 q_2 t_1 q_2^{-1} + t_2 \tag{5}$$

- Работает только для <u>изотропных</u> масштабирований!

Анимация трёхмерных моделей

- Анимация положения объекта в пространстве – неплохо, но скучно

Анимация трёхмерных моделей

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

Анимация трёхмерных моделей

- 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*)

Покадровая анимация моделей

- 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)

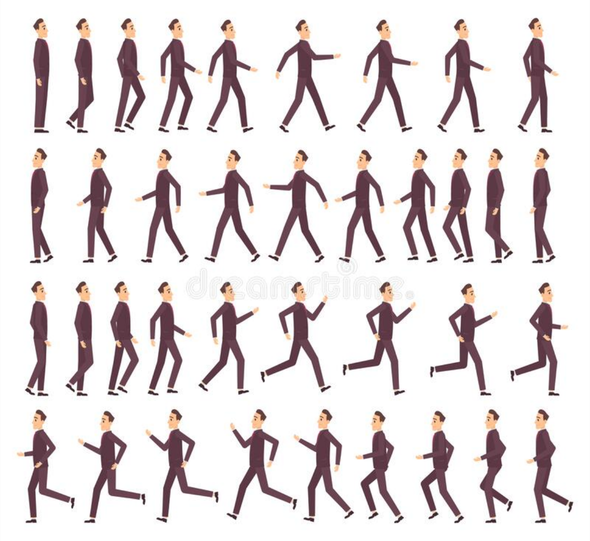
Покадровая анимация моделей

- 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

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
- 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)
    - — Требует 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)
  - — Требует 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)
  - — Требует 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)

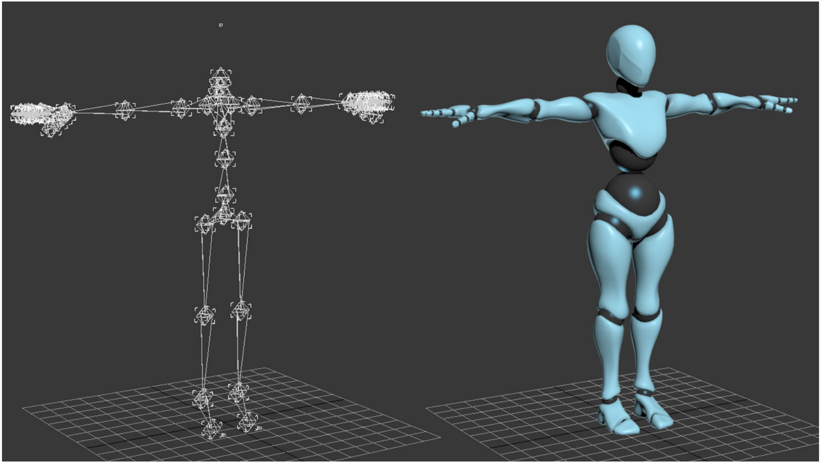
Скелетная анимация моделей

- 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 $\Longrightarrow$ 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;
}
```

Скелетная анимация моделей

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

Скелетная анимация моделей

- К нормалям тоже нужно применять преобразования (но не сдвиги!)
- Преобразования костей часто заданы в локальной для кости системе координат для удобства $\Longrightarrow$ 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 нормалям тоже нужно применять преобразования (но не сдвиги!)
- 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`)
- Кости обычно тоже выстроены в иерархию $\implies$ перед применением нужно вычислить суммарное преобразование (композицию)

Скелетная анимация моделей

- Преобразования для кости и для родительской кости заданы в разных локальных системах координат $\Longrightarrow$ в композицию преобразований нужно вставить смену системы координат

Скелетная анимация моделей

- Преобразования для кости и для родительской кости заданы в разных локальных системах координат $\Longrightarrow$ в композицию преобразований нужно вставить смену системы координат
- В формате glTF считается, что смена системы координат от дочерней к родительской кости *встроена в саму анимацию*

Скелетная анимация моделей

- Преобразования для кости и для родительской кости заданы в разных локальных системах координат $\implies$ в композицию преобразований нужно вставить смену системы координат
- В формате glTF считается, что смена системы координат от дочерней к родительской кости *встроена в саму анимацию*
- $\implies$ нужно откуда-то взять преобразования для костей, если они не описаны в анимации (в glTF это исходные преобразования соответствующих нод)

Скелетная анимация моделей

- Преобразования для кости и для родительской кости заданы в разных локальных системах координат $\Longrightarrow$ в композицию преобразований нужно вставить смену системы координат
- В формате glTF считается, что смена системы координат от дочерней к родительской кости *встроена в саму анимацию*
- $\Longrightarrow$ нужно откуда-то взять преобразования для костей, если они не описаны в анимации (в glTF это исходные преобразования соответствующих нод)
- Нужно быть внимательным к особенностям задания преобразований и системам координат в разных редакторах и форматах!

Скелетная анимация моделей

- В формате glTF вычисление преобразования для кости выглядит как

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

Скелетная анимация моделей

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

Скелетная анимация моделей

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

Скелетная анимация моделей

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

Скелетная анимация моделей

- + 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

Скелетная анимация

- Откуда брать скелетную анимацию?

Скелетная анимация

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

Скелетная анимация

- 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

Inverse kinematics

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

Inverse 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

Inverse 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
    - Повернуть голову, чтобы она смотрела в нужном направлении
    - Поставить ногу на ландшафт с неизвестным заранее наклоном

Inverse 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
    - Повернуть голову, чтобы она смотрела в нужном направлении
    - Поставить ногу на ландшафт с неизвестным заранее наклоном
    - Повернуть руку так, чтобы кисть схватила нужный объект

Inverse 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
  - 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

Inverse 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
    - 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)*

Inverse kinematics

- Финальное положение точки – функция от преобразований отдельных костей

Inverse kinematics

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

Inverse kinematics

- 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 $p = f(\theta_1, \theta_2, \dots \theta_n)$

Inverse kinematics

- 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 $\implies$ IK сводится к задаче решения нелинейной системы уравнений

Inverse kinematics

- 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 $p = f(\theta_1, \theta_2, \ldots \theta_n)$
- p известна, $\{\theta_i\}$ неизвестны $\implies$ IK сводится к задаче решения нелинейной системы уравнений
- В некоторых простых частных случаях можно решить явно (напр. поровот головы – `atan2` от вектора направления)

Inverse kinematics

- 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 $p = f(\theta_1, \theta_2, \ldots \theta_n)$
- p известна, $\{\theta_i\}$ неизвестны $\Longrightarrow$ IK 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

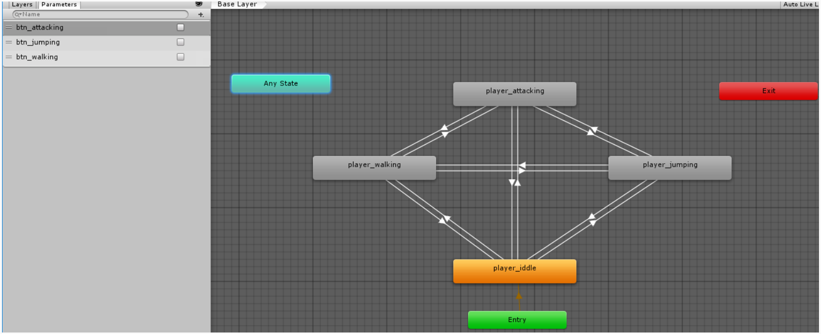
- Если для модели задано несколько анимаций (ходьба, бег, прыжок, поворот, и т.д.) нужно уметь между ними переключаться
- Обычно достаточно интерполировать `local_transform` каждой кости от первой ко второй анимации

- 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) $\Longrightarrow$ часто используют state-машины анимаций с настраиваемыми переходами между состояниями

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) $\Longrightarrow$ 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



Скелетная анимация

- Есть способы для некоторых ситуаций полностью процедурно генерировать анимацию

Скелетная анимация

- 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`

Скелетная анимация: ссылки

- 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-туториал по скелетной анимации
- Видео про графы анимаций в крупных движках

Форматы 3D моделей

- Существует колоссально много форматов для 3D-моделей

Форматы 3D моделей

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

Форматы 3D моделей

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

Форматы 3D моделей

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

Форматы 3D моделей

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

Форматы 3D моделей

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

Форматы 3D моделей

- `.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

Форматы 3D моделей: ссылки

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