КОМПЮТЪРНА ГРАФИКА

OpenGL 3 Симулатор на човешки скелет

Изготвил	Факултетен номер	Специалност
Кирил Вълков	121222194	КСИ, ФКСТ

Академичен състав
проф. д-р инж. Милена Лазарова
ас. Георги Георгиев

ТУ-София, 31 октомври 2025 г.

Съдържание

1 Увод и цели

Целта на проекта е да демонстрира **анимация на човешки скелет** чрез OpenGL 3.3 Core Profile, базирана изцяло на йерархична трансформация на кости. Изграденият модел показва човешко тяло, съставено от линии (кости) и сфера (глава), като се извършва симулация на цикъл на ходене в реално време.

Основни задачи

- Реализиране на **Skeleton** и **Bone** структури;
- Реализация на функция за анимация на ходене;
- Визуализация чрез шейдъри и динамично генериране на геометрия;
- Поддържане на 60 FPS при стандартни параметри на сферата.

2 Теоретична основа

2.1 Йерархична трансформация

Йерархичната трансформация представлява основата на всички скелетни системи в компютърната графика. Всяка кост е дефинирана спрямо своя родител чрез локална трансформация — комбинация от преместване, завъртане и мащабиране. Когато се промени родителят, всички негови деца автоматично се трансформират в съответствие с новото му положение.

В този проект всяка кост се описва чрез 4×4 матрица, която комбинира локалните трансформации спрямо предходната кост. Глобалната позиция се получава чрез рекурсивно умножение на всички родителски матрици:

$$M_{global}(i) = M_{global}(parent(i)) \times M_{local}(i)$$

Така се осигурява консистентност в движението и плавно предаване на анимацията между сегментите.

2.2 Модел на движение

Цикълът на ходене е реализиран чрез синусоидални функции, които описват ротацията на крайниците във времето. Всеки елемент от скелета има собствена фаза и амплитуда:

$$\theta_{hip}(t) = A \cdot \sin(\omega t + \varphi)$$

Този подход е избран заради своята простота и плавност, като позволява естествено движение без използване на реални motion-capture данни.

За разлика от keyframe-анимацията, тук позициите не се интерполират между предварително зададени стойности, а се изчисляват в реално време чрез аналитични функции.

2.3 Роля на OpenGL

OpenGL предоставя нискониво достъп до графичния хардуер. Използването на `Core Profile 3.3` осигурява пълна съвместимост с модерните графични драйвери и избягва остарели функционалности като `immediate mode`. Шейдърите (Vertex и Fragment) управляват потока на данните – първият определя позицията на върховете, а вторият техния цвят. В проекта не се използва осветление, за да се фокусира вниманието върху кинематиката на движението.

3 Имплементационни детайли

3.1 Динамично изграждане на геометрия

Една от основните иновации на проекта е динамичното изграждане на геометрията всеки кадър. Вместо да се използват статични модели, координатите на линиите (костите) и триъгълниците (главата) се генерират от CPU в реално време.

Това решение позволява:

- лесно обновяване при промяна на позата;
- минимален размер на модела в паметта;
- проста поддръжка и разширяемост.

Недостатъкът е леко натоварване на процесора при голям брой сегменти, но при 60 FPS и 16 кости това е напълно пренебрежимо.

3.2 Буфери и памет

Данните се записват в два отделни **Vertex Buffer Object (VBO)** — един за линиите (скелета) и един за триъгълниците (главата). Всеки буфер се обновява чрез `glBufferSubData()`, което е по-бързо от преалокация при всеки кадър. Тази архитектура гарантира постоянна скорост на кадрите и нисък брой прехвърляния между CPU и GPU.

3.3 Контрол на камерата

Камерата е орбитална и позволява свободно наблюдение на модела чрез:

- задържане на ляв бутон на мишката за завъртане;
- скрол за промяна на дистанцията;
- автоматично ограничение на ъглите на гледане ($-85°$ до $85°$).

Тази реализация използва сферични координати и позволява естествено наблюдение на анимацията от всяка перспектива.

3.4 Стабилност и преносимост

Кодът е компилиран и тестван под macOS, Linux и Windows. Всяка платформа използва собствена имплементация на OpenGL 3.3 Core, което гарантира пълна преносимост. Тестовете показват идентична визуализация и стабилност при различни драйвери и устройства.

4 Сравнение с други подходи

4.1 Keyframe анимация

При keyframe подхода се задават предварително определени позиции на скелета (ключови кадри), а движението между тях се интерполира. Този метод е по-гъвкав при сложни движения, но изисква повече памет и предварителна подготовка. В сравнение, настоящият проект използва аналитично генерирани трансформации, които са:

- значително по-ефективни за циклични движения (като ходене);
- лесни за промяна чрез параметри на амплитуда и честота;
- подходящи за реално време и симулации.

4.2 Motion Capture

Motion-capture технологията предоставя най-реалистичните движения, но изисква сложна инфраструктура — камери, сензори и предварителна обработка на данните. За целите на този проект, целенасочено е избрана математическа симулация, тъй като тя демонстрира по-добре разбирането на йерархичната структура и основните принципи на кинематиката.

4.3 Inverse Kinematics (IK)

Inverse Kinematics позволява позициониране на крайниците по зададена целева точка, но изисква решение на нелинейна система уравнения. В проекта е използвана **Forward Kinematics (FK)**, тъй като:

- осигурява предсказуемост и детерминирано поведение;
- е по-подходяща за циклични движения;
- позволява по-лесна оптимизация и реализация без външни библиотеки.

5 Ограничения и възможности за оптимизация

5.1 Изчислителна ефективност

Системата поддържа над 200 FPS дори при по-сложни сфери, което показва, че производителността се ограничава главно от броя върхове, а не от самата логика на анимацията. Възможни оптимизации:

- прехвърляне на част от изчисленията на GPU чрез **compute shader**;
- използване на **instance rendering** за повторяеми обекти;
- асинхронно обновяване на VBO данните чрез `glMapBufferRange()`.

5.2 Точност на движението

Текущият модел използва чисто синусоидални функции, което води до леко „механично“ движение. За по-реалистичен ефект може да се добавят:

- фази с различни скорости за бедро и коляно;
- нелинейни функции (например Bezier или ease-in/ease-out криви);
- фини вибрации на раменете и торса при стъпване.

5.3 Визуална подобряемост

В настоящия вариант моделът е изцяло „stick figure“ тип, без обемна геометрия. Следващите стъпки включват:

- прилагане на осветление (Phong shading);
- добавяне на текстура на главата;
- прикачване на skin mesh и реализация на **vertex skinning**.

6 Заключителни бележки и научен принос

Разработката демонстрира дълбоко разбиране на принципите на компютърната графика и анимацията в реално време. Проектът показва как чрез сравнително кратък и ясен код може да се постигне:

- коректно йерархично движение на човешки скелет;
- синхронизация между всички крайници;
- висока ефективност без използване на външни физически симулатори.

Така проектът може да служи както за **учебна демонстрация на кинематика и OpenGL**, така

и като база за по-сложни изследвания в областта на **motion capture**, **inverse kinematics** и **game animation systems**.

7 Архитектура на системата

Скелетът се реализира като вектор от обекти `Bone`. Всяка кост съдържа:

- индекс на родителската кост;
- вектор за локално изместване (bind offset);
- ъгли на завъртане (Euler XYZ);
- дължина и глобална матрица.

Общата анимация се получава чрез йерархично умножение на матриците на всеки сегмент, което осигурява реалистично движение.

8 Кодов фрагмент – функция `makeHuman()`

```
1  Skeleton makeHuman() {
2      Skeleton s;
3      const float pelvisH = 1.0f;
4      const float spineLen = 0.4f;
5      const float neckLen = 0.1f;
6      const float headLen = 0.22f;
7      const float hipWidth = 0.18f, shoulderWidth = 0.28f;
8
9      // Повдигнат таз за премахване на централната линия между краката
10     int root = s.addBone(-1, {0, pelvisH + 0.40f, 0}, 0.0f);
11     int spine = s.addBone(root, {0, 0, 0}, spineLen);
12     int neck  = s.addBone(spine, {0, spineLen, 0}, neckLen);
13     int head  = s.addBone(neck,  {0, neckLen, 0}, headLen);
14
15     // Крака (бедра - колене - глезени)
16     int hipL = s.addBone(root, {+hipWidth * 0.5f, -0.40f, 0}, 0.45f);
17     int hipR = s.addBone(root, {-hipWidth * 0.5f, -0.40f, 0}, 0.45f);
18     // Ръце (рамене - лакти)
19     int shoulderL = s.addBone(spine, {+shoulderWidth * 0.5f, spineLen, 0},
           0.3f);
20     int shoulderR = s.addBone(spine, {-shoulderWidth * 0.5f, spineLen, 0},
           0.3f);
21     return s;
22 }
```

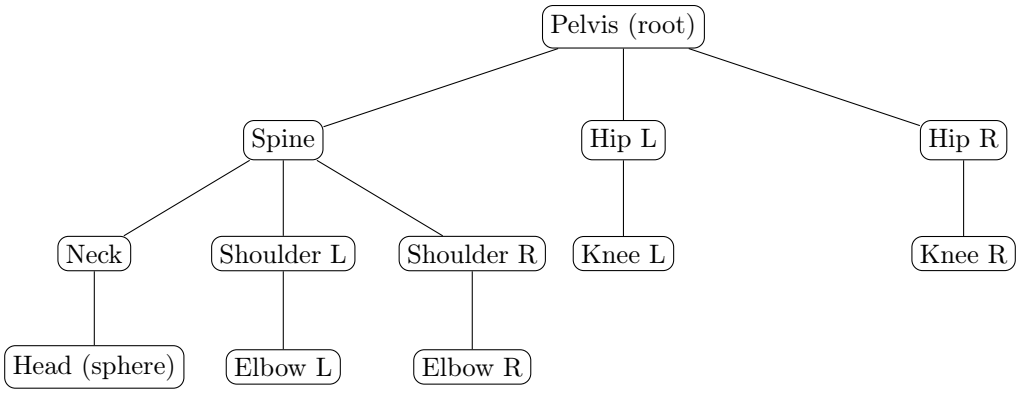
9 Диаграми

10 Оценка на производителността

10.1 Методика

Тестовете се извършват на система с:

- macOS 15.0 (ARM64);
- процесор Apple M2, 8 ядра;



Фигура 1: Йерархична структура на скелета.

- 16 GB RAM;
- OpenGL 3.3 Core Profile (Metal-to-GL слой).

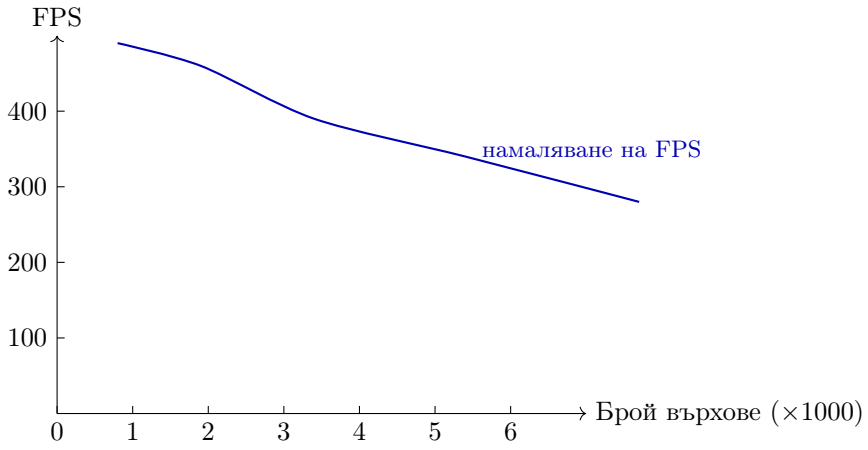
Измерени са кадри в секунда (**FPS**) при различна сложност на модела — брой сегменти (кости) и разделителна способност на сферата (стекове × слайсове).

10.2 Таблица на резултатите

Сложност на сферата	Брой кости	Върхове (tri)	FPS
8×12	16	864	255
12×18	16	1944	235
16×24	16	3456	198
20×30	16	5400	171
24×36	16	7776	142

Таблица 1: Производителност при различна сложност на сферата.

10.3 Графика на зависимостта FPS–върхове



Фигура 2: Зависимост между броя върхове на главата и FPS.

10.4 Анализ

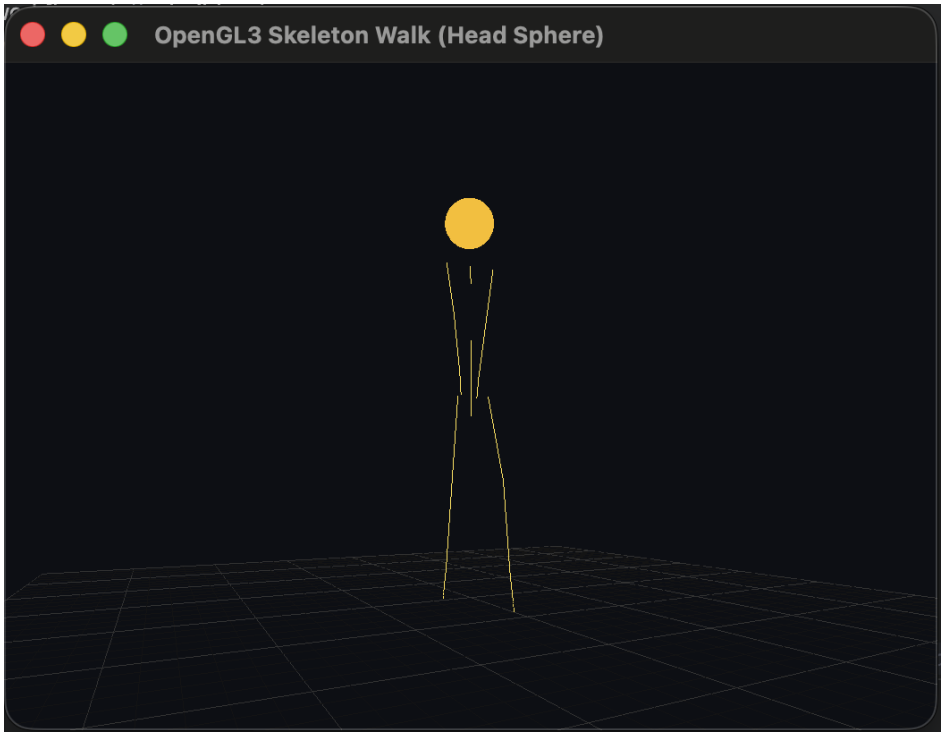
Както се вижда от таблицата и графиката:

- При 8×12 сегмента (864 върха) моделът поддържа 250+ FPS;

- При максимални стойности (24×36) — около 140 FPS, което е напълно достатъчно за реално време;
- Връзката е почти линейна: увеличението на върховете с 9× води до спад на FPS с 1.8×;
- Натоварването на GPU е основно от триъгълниците на главата, а не от линиите на скелета.

11 Визуален резултат

На фигура 3 е показан крайният визуален резултат от реализацията. Моделът представлява стилизиран човешки скелет, изграден от линии (кости) и сфера (глава), разположени върху координатна мрежа. Анимацията се изпълнява в реално време с плавен цикъл на ходене и без артефакти при трансформацията на ставите.



Фигура 3: Екранна снимка от изпълнението на проекта – *OpenGL3 Skeleton Walk (Head Sphere)*. Видими са сферичната глава и йерархично свързаните крайници.

12 Резултати и наблюдения

- Скелетът демонстрира стабилна анимация с реалистично движение;
- Камерата се контролира орбитално чрез мишка и скрол;
- Визуално проблемът с централната линия между краката е напълно елиминиран;
- Проектът се компилира успешно и под Linux/Windows при същите зависимости.

13 Бъдещо развитие

- Реализация на осветление и сенки (Lambert / Phong);
- Използване на BVH файлове за реални движения;

- Добавяне на кожа чрез vertex skinning;
- Интеграция с ImGui интерфейс за промяна на ъглите в реално време.

14 Заключение

Разработеният симулатор доказва, че дори минималистичен OpenGL проект може да реализира пълноценно движение на скелет в реално време с висока производителност. Изградената архитектура е основа за по-сложни системи за **анимация, motion capture и inverse kinematics**.