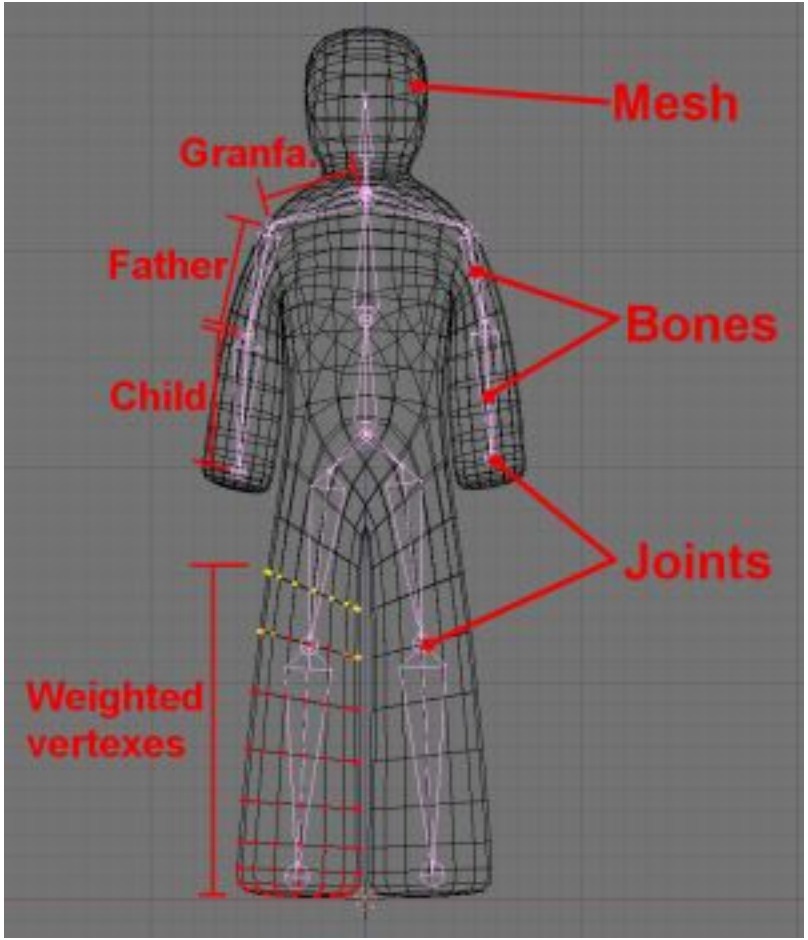
Skeletal animation

Байгельдин Александр, Федоров Роман.

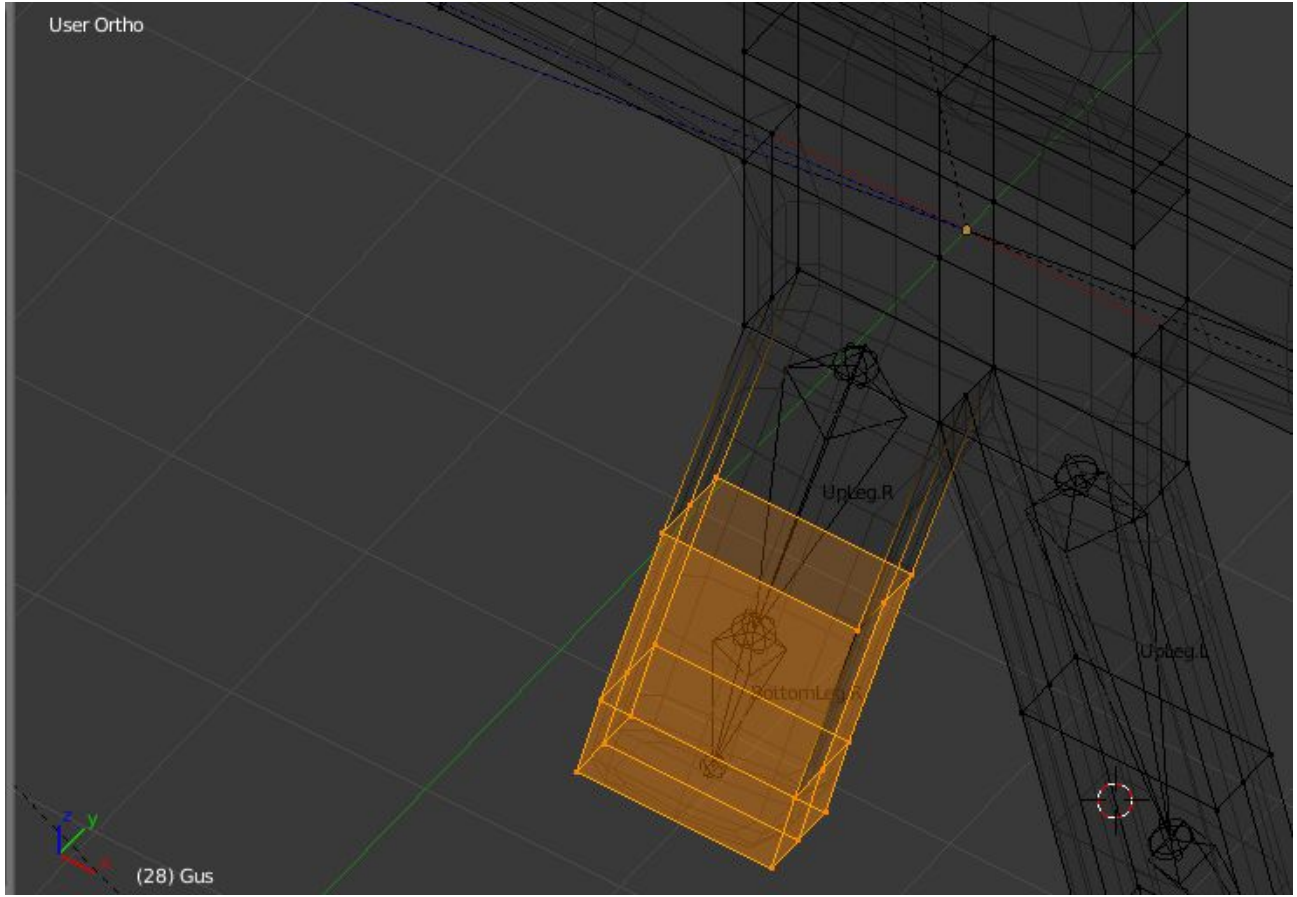
Скелет



Сетка (mesh)

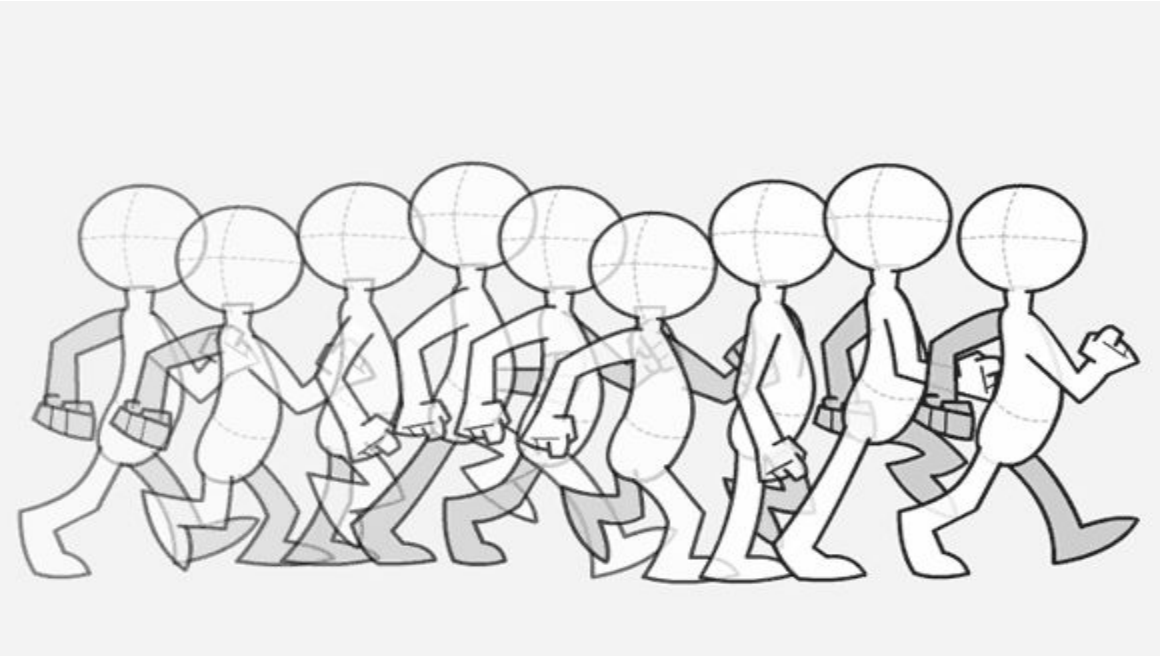
На скелет натянута полигональная сетка (mesh). Каждая вершина сетки соединена с одним или несколькими узлами скелета. Для каждого такого узла определена степень его влияния на положение вершины сетки в пространстве.

$$V' = \sum_{i=0}^{n} W_i M_i V$$



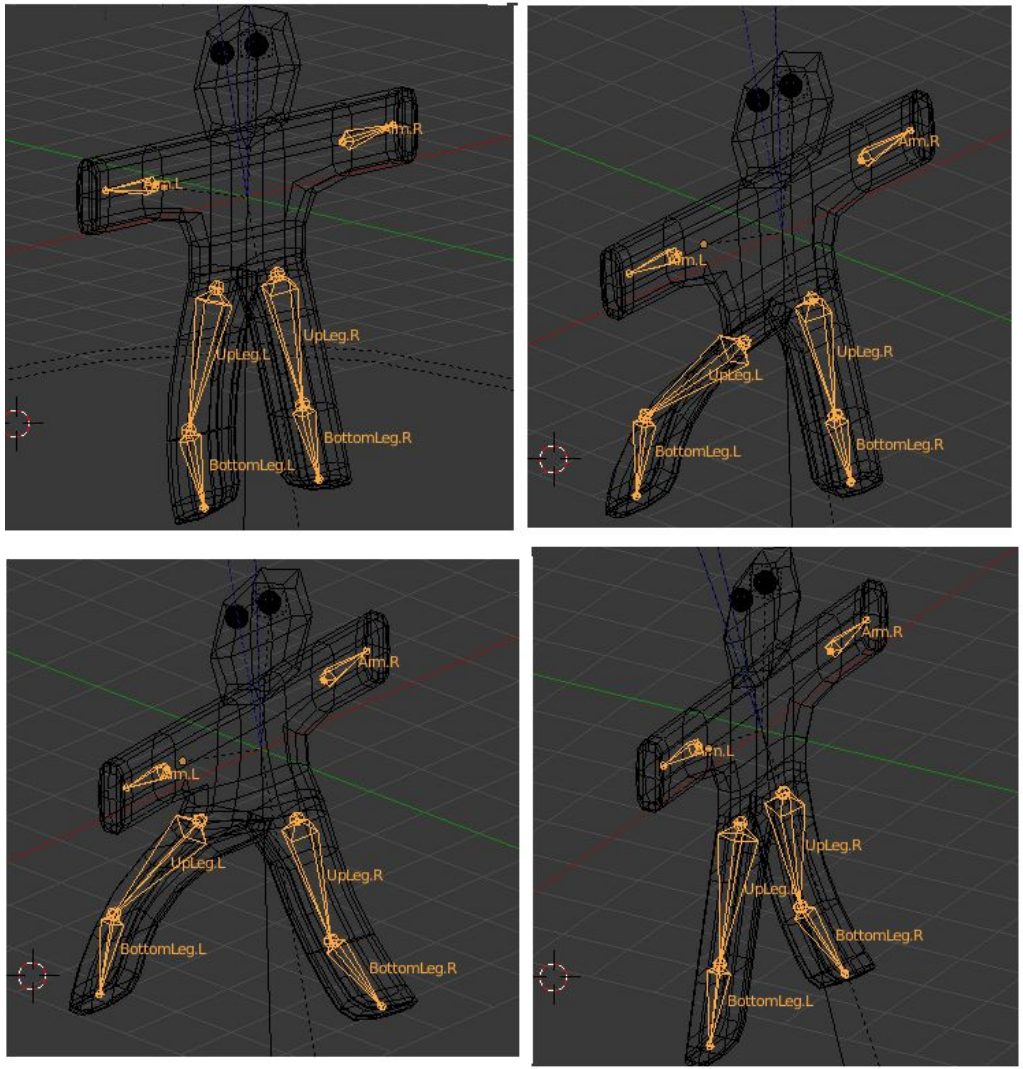
Keyframe animation

Храним полные координаты вершин фигуры для ключевых кадров. Для подсчета промежуточного кадра пытаемся совместить соседние ключевые кадры.



Skeletal animation

Также есть ключевые кадры, но в них содержится положение всех костей (смещение и поворот) в определенный момент. Для вычисления положения фигуры в промежуточный момент совмещаем соседние кадры.



Skeletal vs. Keyframe animation

Keyframe animation:

(+) Простота в реализации.

(+) Не нужно тратить время на подсчет матриц преобазования.

(+) Предсказуемость (т.к. анимация подготовлена заранее).

(-) Накладно по памяти, каждый кадр хранит все вершины.

(-) Неинтерактивно. Реализовать физику почти невозможно.

Skeletal animation:

(+) Занимает мало места.

(+) Интерактивность. Модель может взаимодействовать с миром.

(+) Переиспользуемость. Одинаковые анимации для разных моделей.

(-) Накладно по вычислениям.

Реализация: технологии

- JavaScript
- WebGL API (OpenGL ES 2.0)
- GLSL (OpenGL Shading Language)
- Blender
- gl-matrix

Реализация: задачи

- 3D модель в Blender
- Парсинг JSON модели
- Вершинный и фрагментный шейдеры
- Компиляция шейдеров и модели в WebGL
- Создание интерфейса

Реализация: модель

Three.JS JSON import subset:

- bones (rotation, position, parent)
- animations (frames)
- skin (skin indices, skin weights)
- faces (vertices, material, uvs, normals)
- materials (texture)

Реализация: подсчет матриц

```
Mesh.prototype._adjustBone = function(bone, rot, pos) {
    bone.worldMatrix = mat4.create();
    bone.localMatrix = mat4.create();

    // Считаем локальную матрицу для одной кости
    mat4.fromRotationTranslation(bone.localMatrix, rot, pos);

    if (bone.parent === -1) {
        // Если кость корневая, то значит ее локальная матрица уже в мировых координатах
        mat4.copy(bone.worldMatrix, bone.localMatrix)
    } else {
        // А если нет, то перемножим ее на матрицу родительской кости
        mat4.multiply(bone.worldMatrix,
            this.geometry.bones[bone.parent].worldMatrix, bone.localMatrix)
    }
}
```

Реализация: алгоритм

```
// uP - матрица перспективного отображения
// uM - матрица вида (из координат мира в координаты камеры)
// uM - матрица модели (из координат модели в мировые)
// bt - афинное преобразование, основанное на положениях костей

gl_Position = uP * uV * uM * bt * vec4(aVertex, 1.0);
```

Реализация: особенности

$$V' = W_0 M_0 V + W_1 M_1 V$$

$$V' = \sum_{i=0}^{n} W_i M_i^{-1} M_i V = \sum_{i=0}^{n} W_i I_4 V = 1 I_4 V = V$$

Результат

Спасибо за внимание!