Компьютерная графика
WebGL

Лекция 7

Демяненко Я.М. ЮФУ 2024 MAGnUS

Система частиц

Визуальные эффекты:

- дым
- осадки
- взрывы
- искры
- фейерверки
- бенгальский огонь (рассмотрим)
- и не только

Характеристики системы частиц «бенгальский огонь»

- источник появления частиц (эмиттер)
- каждая частица будет иметь свои:
  - свою скорость
  - направление
  - время жизни
  - будет оставлять за собой след
- но в то же время все частицы подчиняются определённому закону, общему для системы.

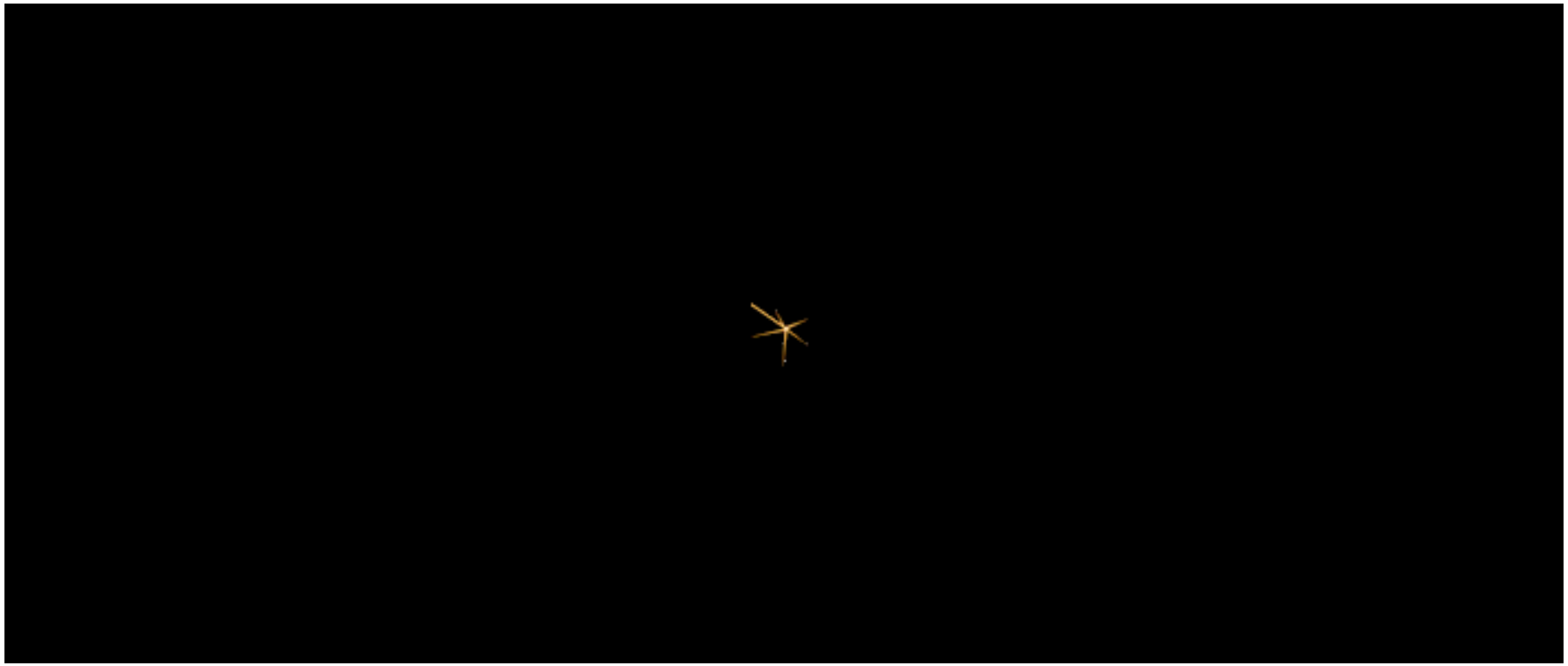
Объекты, используемые для частиц

- полноценный 3D-объект
- 2D-объект с заливкой или текстурой
- точка (спрайт)

Для искр подходит спрайт, поскольку

- для каждой частицы достаточно всего одной вершины
- спрайт по определению всё время повёрнут в сторону наблюдателя, независимо от поворота сцены, и искра всегда остаётся искрой

Одна искра



Одна искра

текстура для точки – изображение с прозрачным фоном 

зададим размер точки с помощью gl_PointSize в вершинном шейдере,
тогда точка превращается в квадрат указанного размера с центром в этой точке

Различия:

- квадрат из двух треугольников будет подвержен трансформациям матриц
- заданный одной точкой квадрат будет всегда направлен к наблюдателю и будет иметь размер gl_PointSize

Желательно, чтобы размер точки совпадал с размером изображения искры

Вершинный шейдер

```
attribute vec3 a_position;

uniform mat4 u_mvMatrix;
uniform mat4 u_pMatrix;

void main() {
    gl_Position = u_pMatrix * u_mvMatrix * vec4(a_position, 1.0);

    // размер искры
    gl_PointSize = 32.0;
}
```

Фрагментный шейдер

```
precision mediump float;

   uniform sampler2D u_texture;

   void main() {
      gl_FragColor = texture2D(u_texture, gl_PointCoord);
   }
```

Загрузка изображения и инициализация текстуры

```
var texture = gl.createTexture();
var image = new Image();
image.crossOrigin = "anonymous";
image.src = "spark.png";

image.addEventListener('load', function() {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.bindTexture(gl.TEXTURE_2D, null);

    // отрисовку сцены начинаем только после загрузки изображения
    requestAnimationFrame(drawScene);
});
```

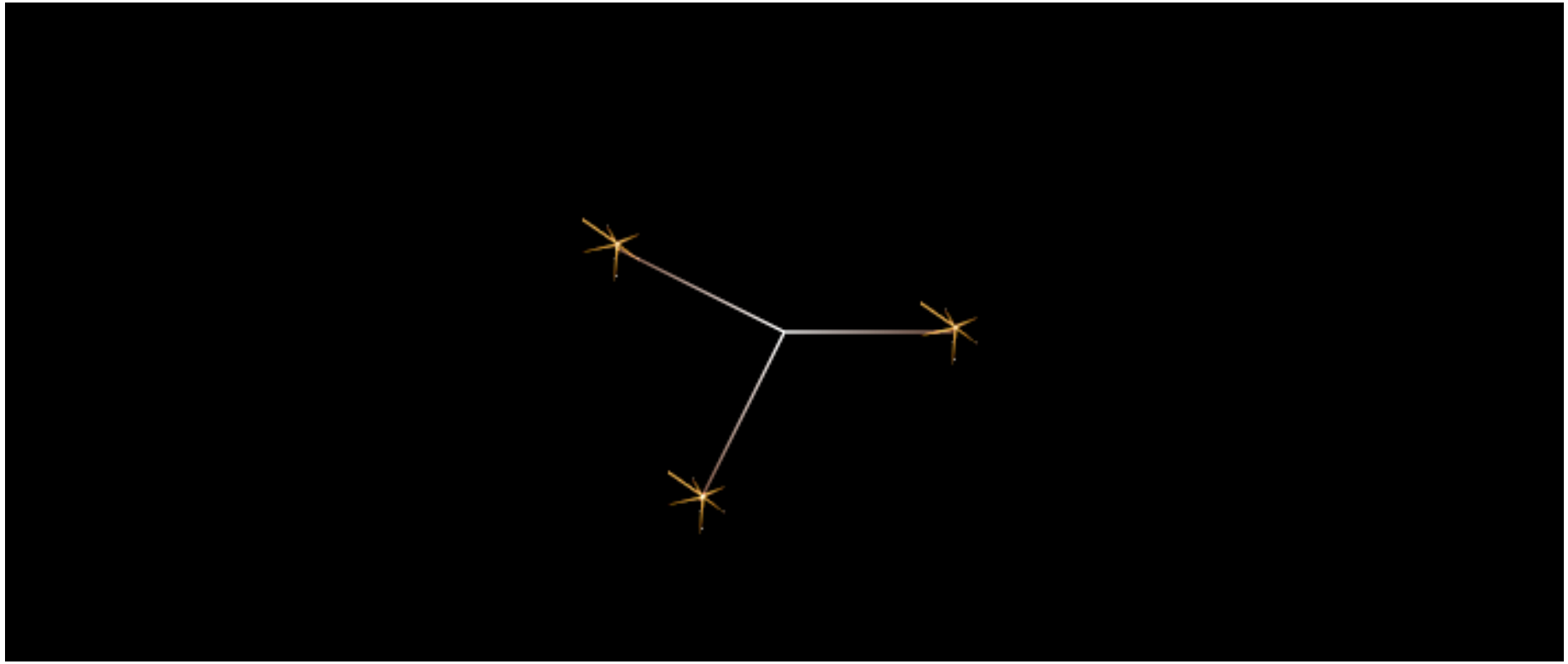
NEAREST LINEAR

Смешивание цветов

Чтобы WebGL понимал прозрачность и избавлялся от фона искры, нам необходимо включить смешивание цветов

```
//включает смешивание
gl.enable(gl.BLEND);

//определяет взаимодействие рисуемых и уже отрисованных пикселей
gl.blendFunc(gl.SRC_ALPHA, gl.ONE);
```

Добавляем следы искр



Ещё один вершинный шейдер

```
attribute vec3 a_position;
attribute vec3 a_color;

out vec3 v_color;

uniform mat4 u_mvMatrix;
uniform mat4 u_pMatrix;

void main() {
    v_color = a_color;
    gl_Position = u_pMatrix * u_mvMatrix * vec4(a_position, 1.0);
}
```

Ещё один фрагментный шейдер

```
precision mediump float;

in vec3 v_color;

void main() {
    gl_FragColor = vec4(v_color, 1.0);
}
```

Инициализация искр

```
//инициализация программы искр

var programSpark = webglUtils.createProgramFromScripts(gl, ["vertex-shader-spark", "fragment-shader-spark"]);

var positionAttributeLocationSpark = gl.getAttribLocation(programSpark, "a_position");
var textureLocationSpark = gl.getUniformLocation(programSpark, "u_texture");
var pMatrixUniformLocationSpark = gl.getUniformLocation(programSpark, "u_pMatrix");
var mvMatrixUniformLocationSpark = gl.getUniformLocation(programSpark, "u_mvMatrix");
```

Инициализация следов искр

```
// инициализация программы следов искр

var programTrack = webglUtils.createProgramFromScripts(gl, ["vertex-shader-track", "fragment-shader-track"]);

 var positionAttributeLocationTrack = gl.getAttribLocation(programTrack, "a_position");
 var colorAttributeLocationTrack = gl.getAttribLocation(programTrack, "a_color");
 var pMatrixUniformLocationTrack = gl.getUniformLocation(programTrack, "u_pMatrix");
 var mvMatrixUniformLocationTrack = gl.getUniformLocation(programTrack, "u_mvMatrix");
```

Отрисовка следов и самих искр

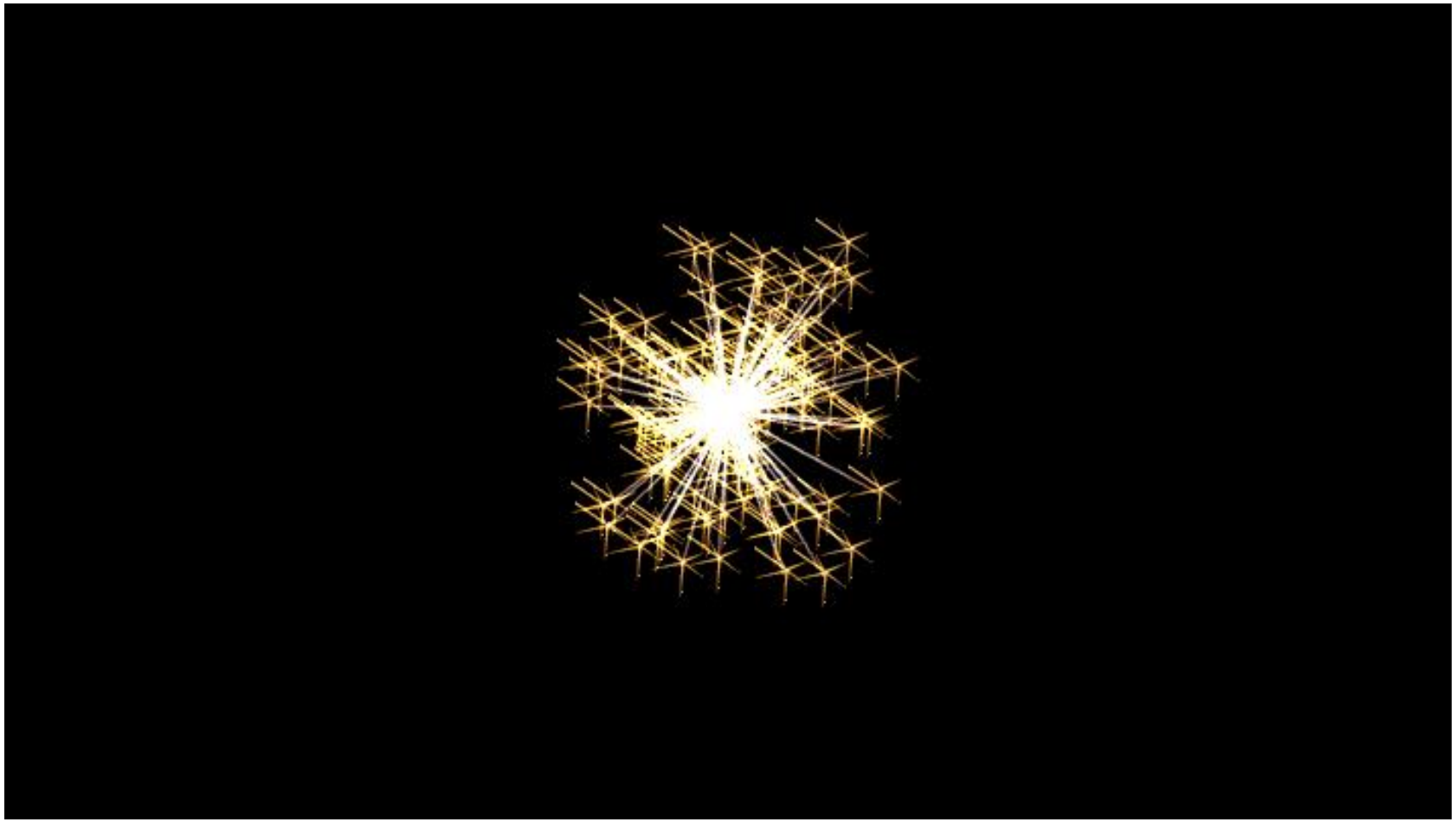
В каждую функцию будут передаваться координаты всех существующих на данный момент искр — сейчас три

```
var positions = [
       1, 0,   0,
      -1, 0.5, 0,
      -0.5, -1, 0
 ];

// В начале каждой функции отрисовки drawTracks и drawSparks
// будет вызываться gl.useProgram для установки текущей программы

 drawTracks(positions);  // будет активироваться программа programTrack
 drawSparks(positions); // будет активироваться программа programSpark
```

Отрисовка следов искр

```
var colors = [];
var positionsFromCenter = [];
for (var i = 0; i < positions.length; i += 3) {
   // для каждой координаты добавляем точку начала координат, чтобы получить след искры
   positionsFromCenter.push(0, 0, 0);
   positionsFromCenter.push(positions[i], positions[i + 1], positions[i + 2]);

   // цвет в начале координат будет белый (горячий), а дальше будет приближаться к оранжевому
   colors.push(1, 1, 1, 0.47, 0.31, 0.24);
 }
```

Полноценная система частиц



Алгоритм работы бенгальского огня

- создаём искру с произвольными
  - направлением
  - скоростью
  - длиной пути

- при каждой отрисовке изменяем положение искры

- когда искра пройдёт весь отрезок пути, запускаем её заново из начала координат

Класс для искры

```
function Spark() {
    this.init();
};

// количество искр
Spark.sparksCount = 200;
```

С двумя функциями:

- функция init вызывается для создания и инициализации искры
- функция move вызывается при каждом цикле отрисовки для приращения координат искры

```
Spark.prototype.init = function() {
    // время создания искры
    this.timeFromCreation = performance.now();

    // задаём направление полёта искры в градусах, от 0 до 360
    var angle = Math.random() * 360;
    // радиус - это расстояние, которое пролетит искра
    var radius = Math.random();
    // отмеряем точки на окружности - максимальные координаты искры
    this.xMax = Math.cos(angle) * radius;
    this.yMax = Math.sin(angle) * radius;

    // dx и dy - приращение искры за вызов отрисовки, то есть её скорость,
    // у каждой искры своя скорость. multiplier подбирается эмпирически
    var multiplier = 125 + Math.random() * 125;
    this.dx = this.xMax / multiplier;
    this.dy = this.yMax / multiplier;

    // Для того, чтобы не все искры начинали движение из начала координат,
    // делаем каждой искре свой отступ, но не более максимальных значений.
    this.x = (this.dx * 1000) % this.xMax;
    this.y = (this.dy * 1000) % this.yMax;
};
```

```
Spark.prototype.move = function(time) {
    // находим разницу между вызовами отрисовки, чтобы анимация работала
    // одинаково на компьютерах разной мощности
    var timeShift = time - this.timeFromCreation;
    this.timeFromCreation = time;

    // приращение зависит от времени между отрисовками
    var speed = timeShift;
    this.x += this.dx * speed;
    this.y += this.dy * speed;

    // если искра достигла конечной точки, запускаем её заново из начала координат
    if (Math.abs(this.x) > Math.abs(this.xMax) || Math.abs(this.y) > Math.abs(this.yMax)) {
        this.init();
        return;
    }
};
```

Создаем необходимое количество искр при инициализации программы

```
var sparks = [];
  for (var i = 0; i < Spark.sparksCount; i++) {
     sparks.push(new Spark());
  }
```

```
//Вызываем смещение искр при каждой отрисовке
for (var i = 0; i < sparks.length; i++) {
        sparks[i].move(now);
    }

//получаем координаты искр для передачи в функции
var positions = [];
 sparks.forEach(function(item, i, arr) {
        positions.push(item.x);
        positions.push(item.y);
        // искры двигаются только в одной плоскости xy
        positions.push(0);
 });

 drawTracks(positions);
 drawSparks(positions);
```