Глаз на сферу натяну: Воспроизведение сферического видео в браузере

Школенко Виталий

Панорама

Панорамным можно назвать изображение, перекрывающее человеческое поле зрения в пределах 160° по горизонтали и 75° по вертикали. В некоторых случаях обзор может достигать 360° по горизонтали и 180° по вертикали – такие панорамы называются сферическими.

Панорама позволяет создать эффект присутствия, охватить взглядом большое количество деталей.

Панорамная живопись

Часть панорамы «Ночной пир Хань Сицая», XII век, Империя Сун

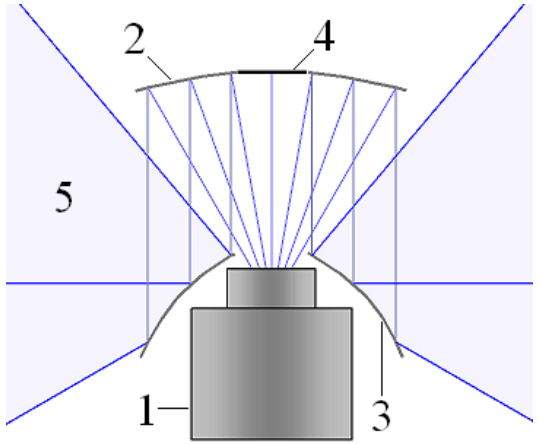
Франц Алексеевич Рубо, полотно панорамы «Бородинская битва» (1912)

Панорамная фотография (склейка)

+ Ps

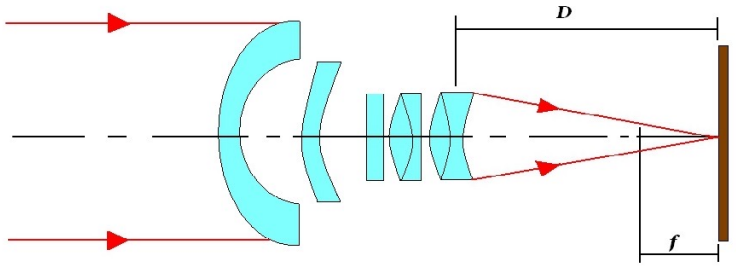
Другие способы снять панораму

Поворотный фотоаппарат Leme (1962г)

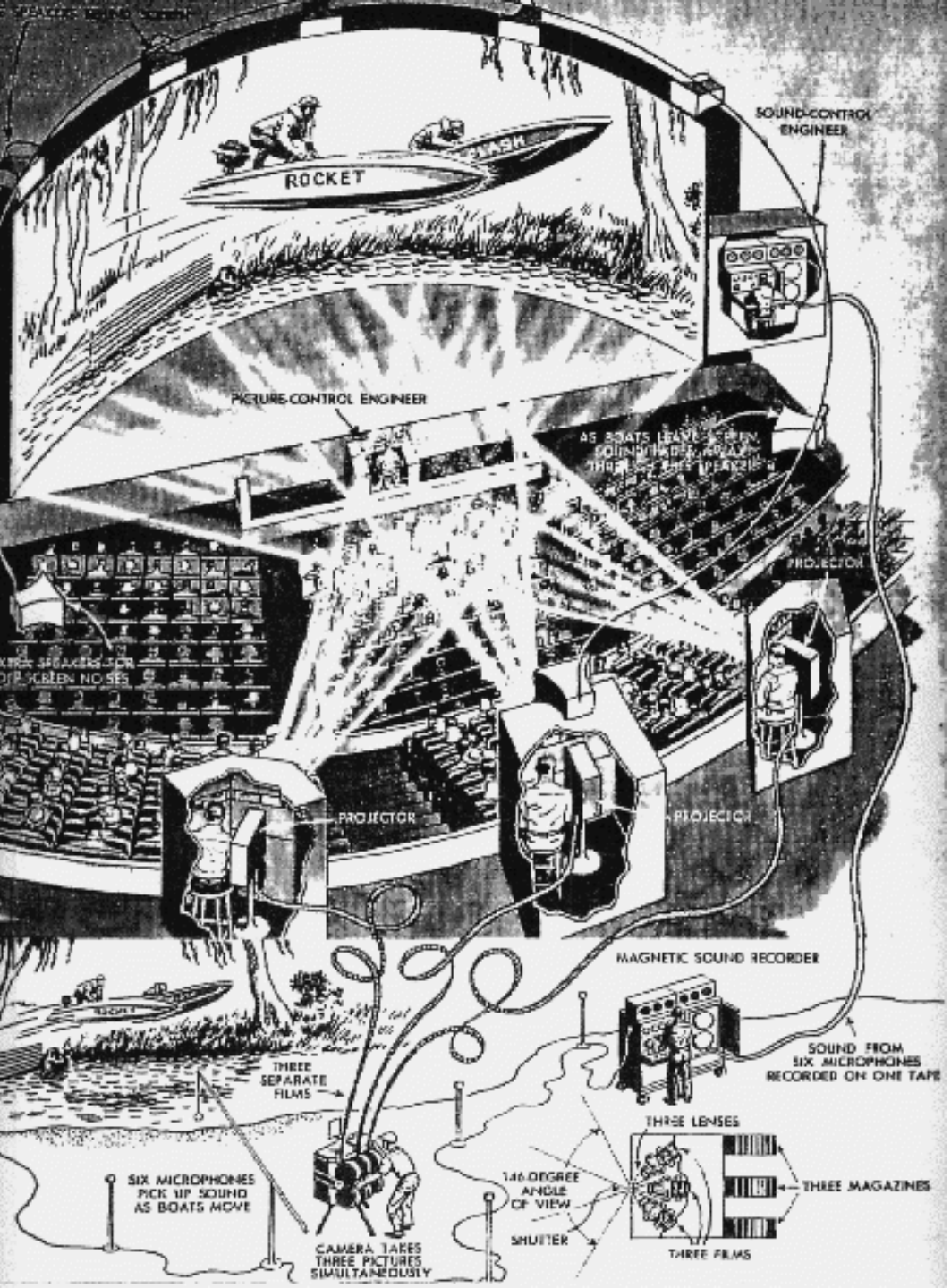


Катадиоптрические системы (начиная с 1814г)

Горизонт S3 U500 (196х – 2004гг)



Объектив «рыбий глаз» придуман Джеймсом Максвеллом в 1858г



Панорамный кинотеатр

Съемку вели несколькими камерами и потом по такой же схеме проецировали на сильно изогнутый экран с нескольких проекторов – своего рода склейка для видеосъемки.

Оборудование для съемки

–2%

Экшн-камера Insta360 ONE X2, черный

С картой Плюса

35362 ₽ / без: 36084 ₽

Экшн-камера GoPro Max CHDHZ-202-RX

44990 ₽

Самовывоз завтра

ВАМ –5%

Панорамная камера VR 360 RICOH THETA SC2 (розовая)

С картой Плюса

49837 ₽ / без: 50854 ₽

–2%

Панорамная камера Ricoh Theta Z1

С картой Плюса

132290 ₽ / без: 134990 ₽

ВАМ –5%

Панорамная камера VR 360 RICOH THETA Z1 51GB

С картой Плюса

171379 ₽ / без: 174379 ₽

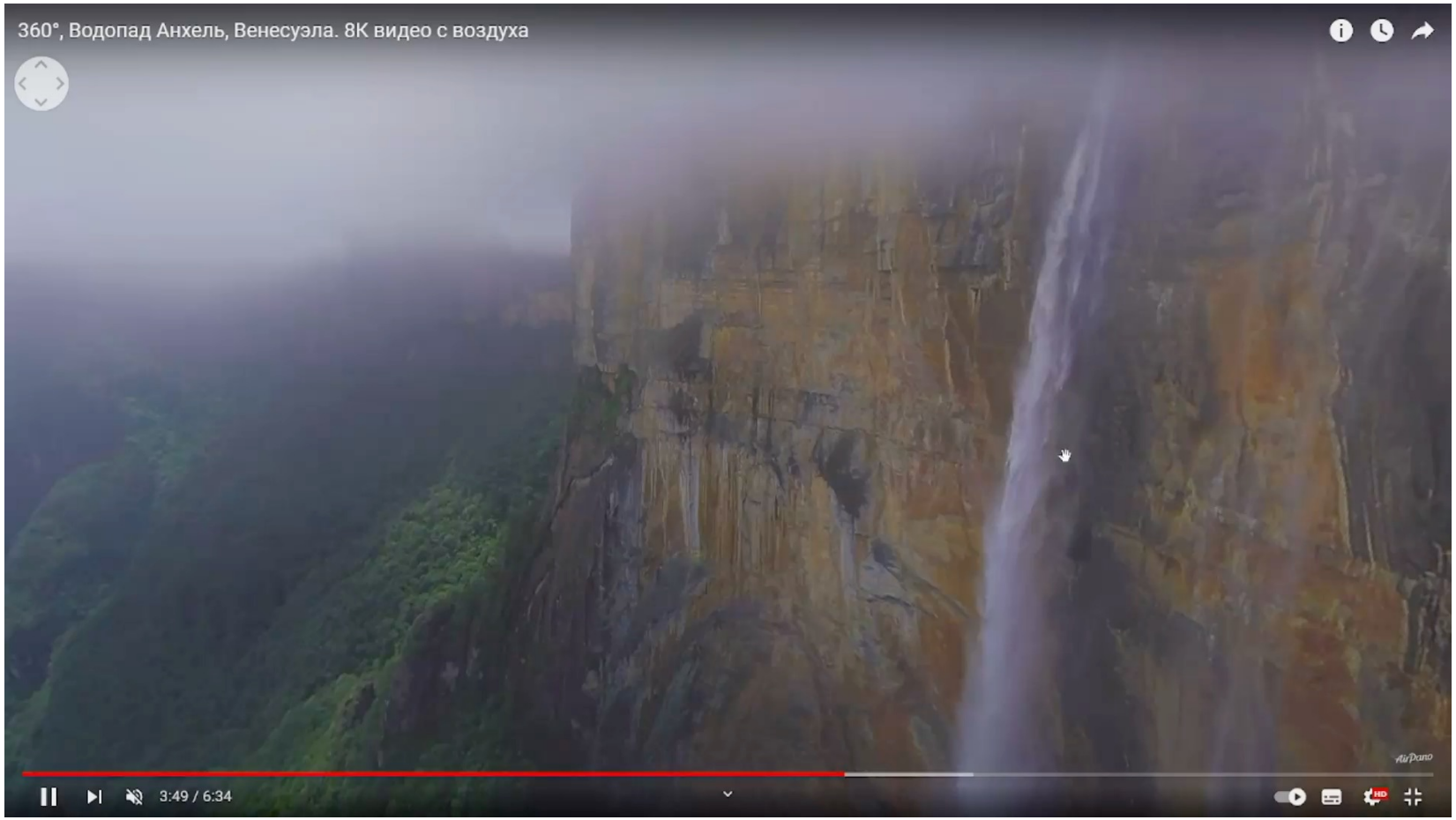
Программное обеспечение

Для работы с панорамным видео

Плееры

Редакторы

Youtube показывает панорамы



https://www.youtube.com/watch?v=L_tqK4eqelA

Жизненный цикл панорамного видео

1. Съемка специальным устройством
2. Упаковка в файл

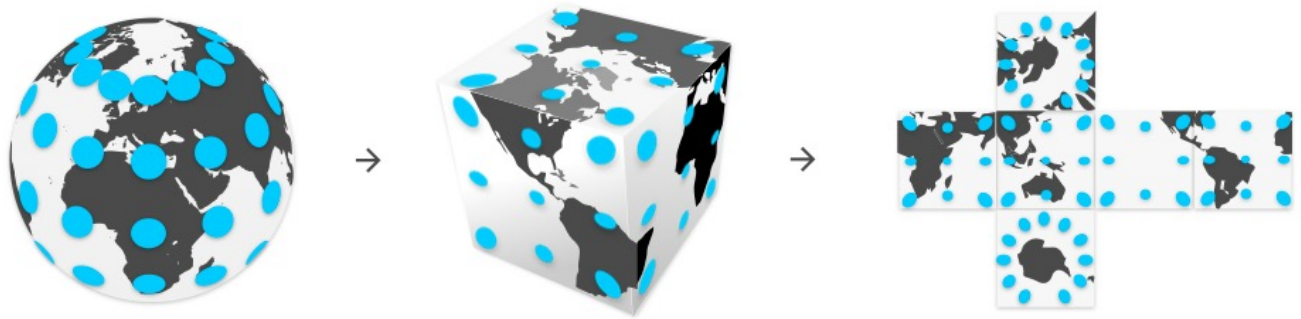
---

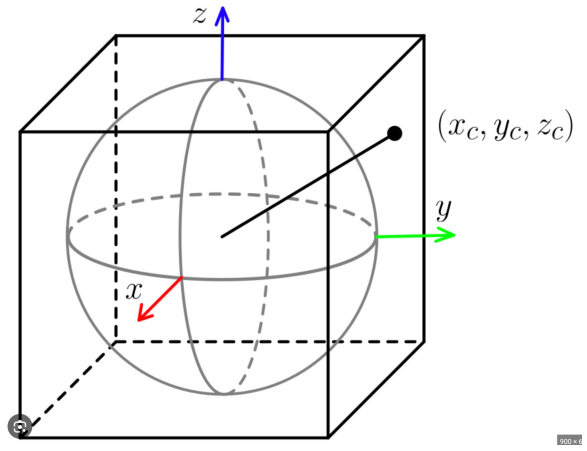
3. Передача/хранение/перекодирование

---

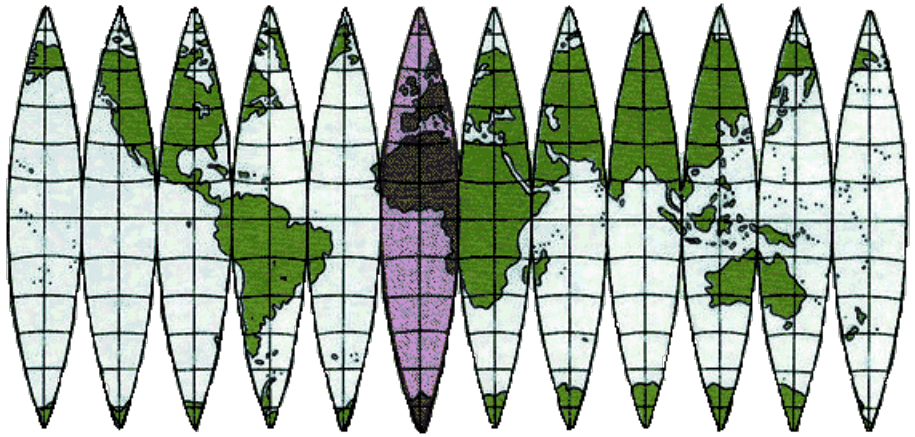
4. Распаковка файла
5. Обратная проекция для получения плоского изображения

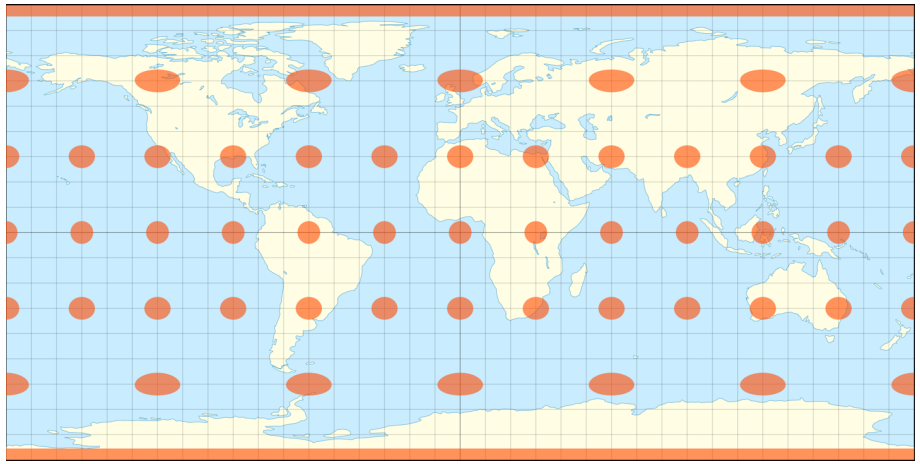
Кубическая проекция





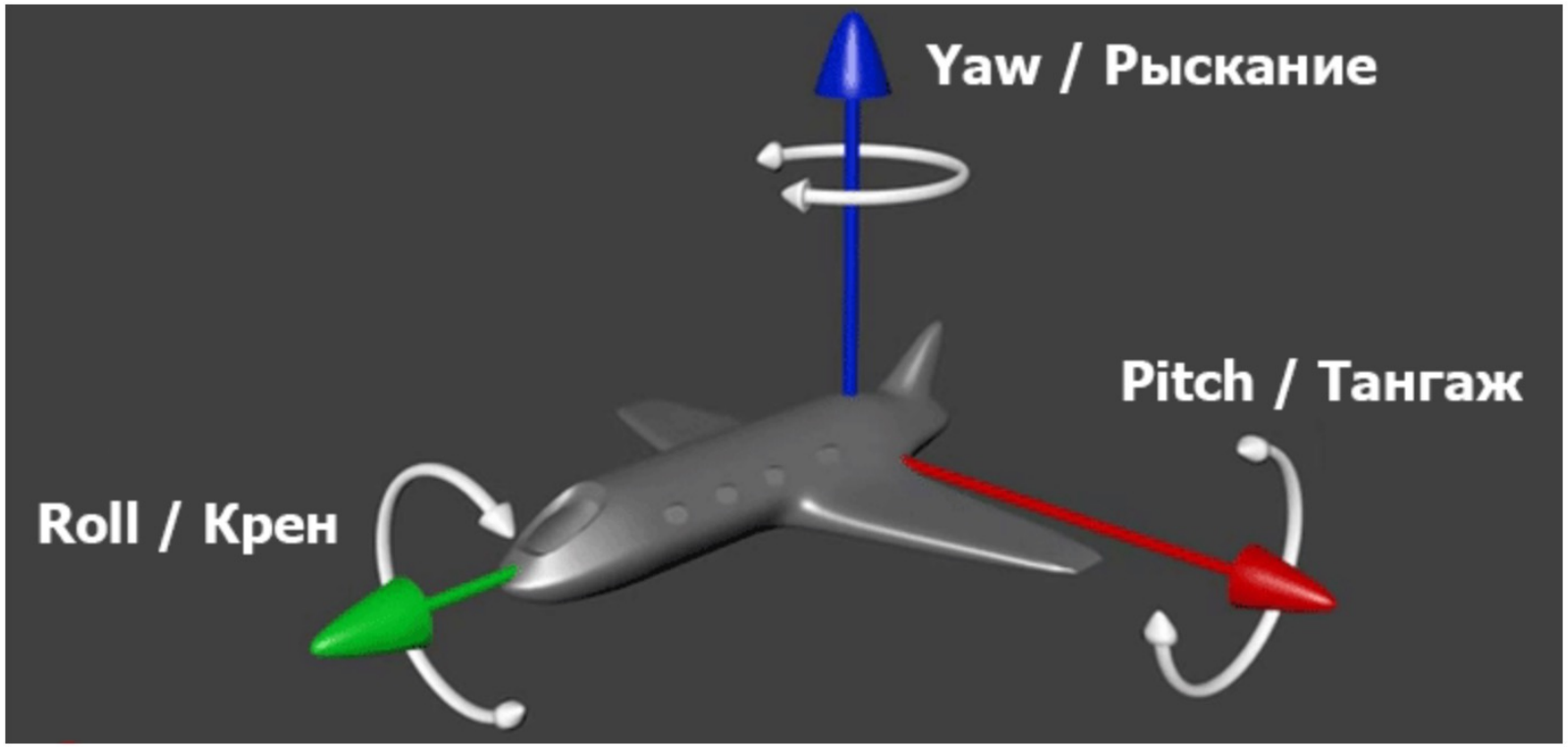
Эквидистантная Проекция





Ориентация камеры

Углы наклона камеры (самолетные, корабельные) – по факту углы Эйлера



Мета-информация в контейнере MP4

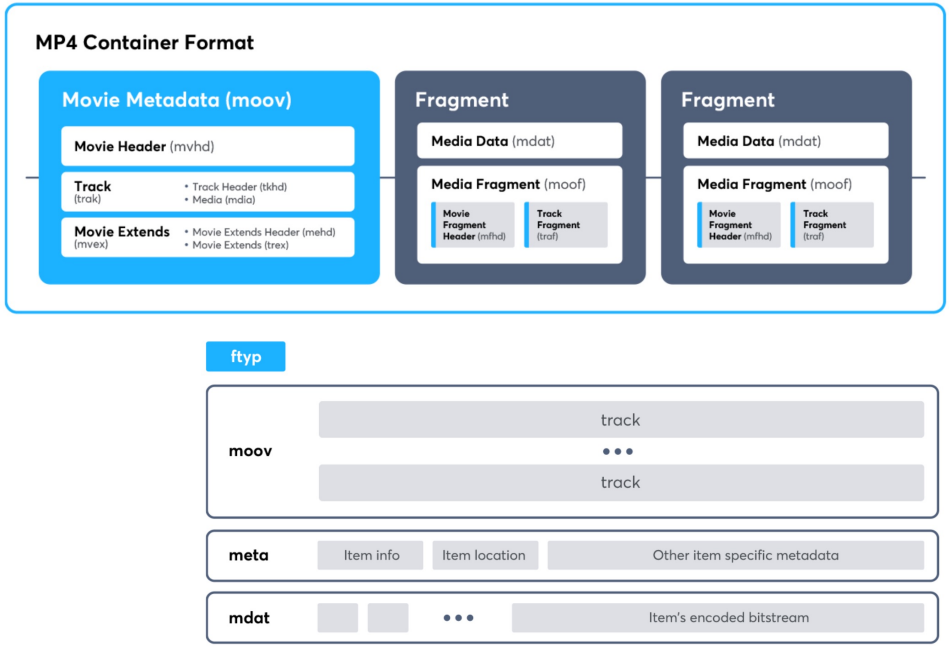
```
aligned(8) class ProjectionHeader extends FullBox('prhd', 0, 0) {
    int(32) pose_yaw_degrees;
    int(32) pose_pitch_degrees;
    int(32) pose_roll_degrees;
}
```

```
aligned(8) class Projection extends Box('proj') {
}
```

```
aligned(8) class CubemapProjection ProjectionDataBox('cbmp', 0, 0) {
    unsigned int(32) layout;
    unsigned int(32) padding;
}
```

right face	left face	up face
down face	front face	back face

```
aligned(8) class EquirectangularProjection ProjectionDataBox('equi', 0, 0) {
    unsigned int(32) projection_bounds_top;
    unsigned int(32) projection_bounds_bottom;
    unsigned int(32) projection_bounds_left;
    unsigned int(32) projection_bounds_right;
}
```



Мета-информация в контейнере Matroska

ProjectionType (0x7671) – Тип проекции

ProjectionPrivate (0x7672) – Доп. Информация по проекции

ProjectionPoseYaw (0x7673) - Угол рысканья камеры

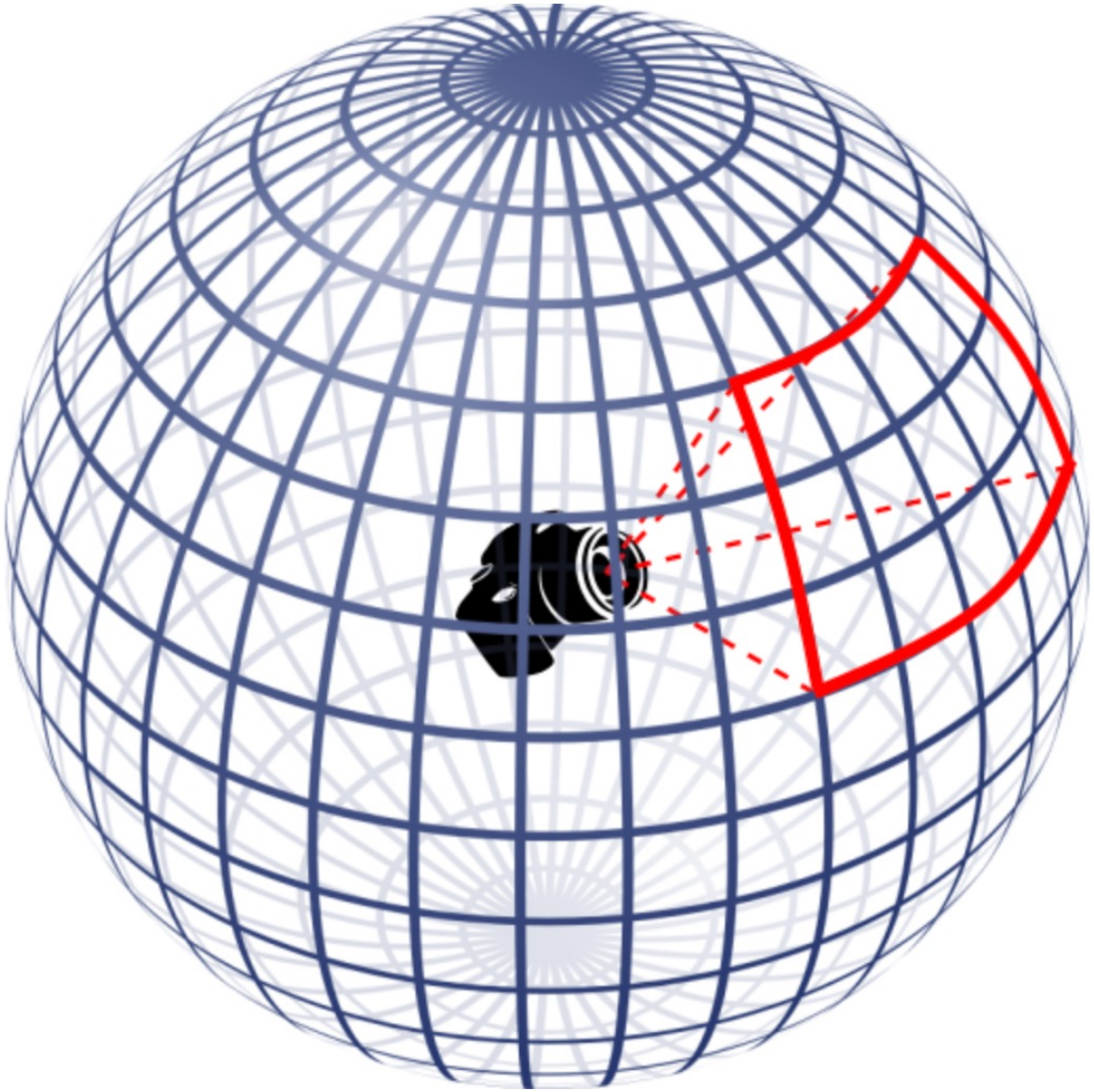
ProjectionPosePitch (0x7674) – Угол тангажа камеры

ProjectionPoseRoll (0x7675) – Угол крена камеры

Сферическое видео без обработки

https://vimeo.com/214403336

Схема решения



Преобразование видео на three.js

```
/** Создаем рендерер */
const renderer = new THREE.WebGLRenderer();
renderer.outputColorSpace = THREE.LinearSRGBColorSpace;
renderer.setSize(CANVAS_WIDTH, CANVAS_HEIGHT);
const target = document.getElementById( elementId: 'result');
target.appendChild(renderer.domElement);

// Создаем камеру
const camera = new THREE.PerspectiveCamera(45, CANVAS_WIDTH / CANVAS_HEIGHT, 1, 300);
camera.position.set(0, 0, 0);
camera.lookAt(100, 0, 0);

// Создаем текстуру на основе видеоэлемента
const texture = new THREE.VideoTexture(videoElement);
// указываем, что текстуру надо обновлять
texture.needsUpdate = true;
texture.wrapS = THREE.RepeatWrapping;
texture.repeat.x = -1;
```

```
// Создаем геометрию
const geometry = new THREE.SphereGeometry(200, 100, 100);
// Создаем материал
const material = new THREE.MeshBasicMaterial({map: texture, side: THREE.BackSide});
material.needsUpdate = true;
// Создаем объект
const mesh = new THREE.Mesh(geometry, material);

// Создаем сцену
const scene = new THREE.Scene();
scene.add(mesh);

renderScene(renderer, scene, camera);
```

Преобразование видео на three.js

```
/** Создаем рендерер */
const renderer = new THREE.WebGLRenderer();
renderer.outputColorSpace = THREE.LinearSRGBColorSpace;
renderer.setSize(CANVAS_WIDTH, CANVAS_HEIGHT);
const target = document.getElementById( elementId: 'result');
target.appendChild(renderer.domElement);

// Создаем камеру
const camera = new THREE.PerspectiveCamera(45, CANVAS_WIDTH / CANVAS_HEIGHT, 1, 300);
camera.position.set(0, 0, 0);
camera.lookAt(100, 0, 0);

// Создаем текстуру на основе видеоэлемента
const texture = new THREE.VideoTexture(videoElement);
// указываем, что текстуру надо обновлять
texture.needsUpdate = true;
texture.wrapS = THREE.RepeatWrapping;
texture.repeat.x = -1;
```

```
// Создаем геометрию
const geometry = new THREE.SphereGeometry(200, 100, 100);
// Создаем материал
const material = new THREE.MeshBasicMaterial({map: texture, side: THREE.BackSide});
material.needsUpdate = true;
// Создаем объект
const mesh = new THREE.Mesh(geometry, material);

// Создаем сцену
const scene = new THREE.Scene();
scene.add(mesh);

renderScene(renderer, scene, camera);
```

Преобразование видео на three.js

```
/** Создаем рендерер */
const renderer = new THREE.WebGLRenderer();
renderer.outputColorSpace = THREE.LinearSRGBColorSpace;
renderer.setSize(CANVAS_WIDTH, CANVAS_HEIGHT);
const target = document.getElementById( elementId: 'result');
target.appendChild(renderer.domElement);

// Создаем камеру
const camera = new THREE.PerspectiveCamera(45, CANVAS_WIDTH / CANVAS_HEIGHT, 1, 300);
camera.position.set(0, 0, 0);
camera.lookAt(100, 0, 0);

// Создаем текстуру на основе видеоэлемента
const texture = new THREE.VideoTexture(videoElement);
// указываем, что текстуру надо обновлять
texture.needsUpdate = true;
texture.wrapS = THREE.RepeatWrapping;
texture.repeat.x = -1;
```

```
// Создаем геометрию
const geometry = new THREE.SphereGeometry(200, 100, 100);
// Создаем материал
const material = new THREE.MeshBasicMaterial({map: texture, side: THREE.BackSide});
material.needsUpdate = true;
// Создаем объект
const mesh = new THREE.Mesh(geometry, material);

// Создаем сцену
const scene = new THREE.Scene();
scene.add(mesh);

renderScene(renderer, scene, camera);
```

Преобразование видео на three.js

```
/** Создаем рендерер */
const renderer = new THREE.WebGLRenderer();
renderer.outputColorSpace = THREE.LinearSRGBColorSpace;
renderer.setSize(CANVAS_WIDTH, CANVAS_HEIGHT);
const target = document.getElementById( elementId: 'result');
target.appendChild(renderer.domElement);

// Создаем камеру
const camera = new THREE.PerspectiveCamera(45, CANVAS_WIDTH / CANVAS_HEIGHT, 1, 300);
camera.position.set(0, 0, 0);
camera.lookAt(100, 0, 0);

// Создаем текстуру на основе видеоэлемента
const texture = new THREE.VideoTexture(videoElement);
// указываем, что текстуру надо обновлять
texture.needsUpdate = true;
texture.wrapS = THREE.RepeatWrapping;
texture.repeat.x = -1;
```

```
// Создаем геометрию
const geometry = new THREE.SphereGeometry(200, 100, 100);
// Создаем материал
const material = new THREE.MeshBasicMaterial({map: texture, side: THREE.BackSide});
material.needsUpdate = true;
// Создаем объект
const mesh = new THREE.Mesh(geometry, material);

// Создаем сцену
const scene = new THREE.Scene();
scene.add(mesh);

renderScene(renderer, scene, camera);
```

Преобразование видео на three.js

```
/** Создаем рендерер */
const renderer = new THREE.WebGLRenderer();
renderer.outputColorSpace = THREE.LinearSRGBColorSpace;
renderer.setSize(CANVAS_WIDTH, CANVAS_HEIGHT);
const target = document.getElementById( elementId: 'result');
target.appendChild(renderer.domElement);

// Создаем камеру
const camera = new THREE.PerspectiveCamera(45, CANVAS_WIDTH / CANVAS_HEIGHT, 1, 300);
camera.position.set(0, 0, 0);
camera.lookAt(100, 0, 0);

// Создаем текстуру на основе видеоэлемента
const texture = new THREE.VideoTexture(videoElement);
// указываем, что текстуру надо обновлять
texture.needsUpdate = true;
texture.wrapS = THREE.RepeatWrapping;
texture.repeat.x = -1;
```

```
// Создаем геометрию
const geometry = new THREE.SphereGeometry(200, 100, 100);
// Создаем материал
const material = new THREE.MeshBasicMaterial({map: texture, side: THREE.BackSide});
material.needsUpdate = true;
// Создаем объект
const mesh = new THREE.Mesh(geometry, material);

// Создаем сцену
const scene = new THREE.Scene();
scene.add(mesh);

renderScene(renderer, scene, camera);
```

Преобразование видео на three.js

```
/** Создаем рендерер */
const renderer = new THREE.WebGLRenderer();
renderer.outputColorSpace = THREE.LinearSRGBColorSpace;
renderer.setSize(CANVAS_WIDTH, CANVAS_HEIGHT);
const target = document.getElementById( elementId: 'result');
target.appendChild(renderer.domElement);

// Создаем камеру
const camera = new THREE.PerspectiveCamera(45, CANVAS_WIDTH / CANVAS_HEIGHT, 1, 300);
camera.position.set(0, 0, 0);
camera.lookAt(100, 0, 0);

// Создаем текстуру на основе видеоэлемента
const texture = new THREE.VideoTexture(videoElement);
// указываем, что текстуру надо обновлять
texture.needsUpdate = true;
texture.wrapS = THREE.RepeatWrapping;
texture.repeat.x = -1;
```

```
// Создаем геометрию
const geometry = new THREE.SphereGeometry(200, 100, 100);
// Создаем материал
const material = new THREE.MeshBasicMaterial({map: texture, side: THREE.BackSide});
material.needsUpdate = true;
// Создаем объект
const mesh = new THREE.Mesh(geometry, material);

// Создаем сцену
const scene = new THREE.Scene();
scene.add(mesh);

renderScene(renderer, scene, camera);
```

Результат

Приключение на 20 минут

Что движок сделал за нас?

Создал сферу в пространстве

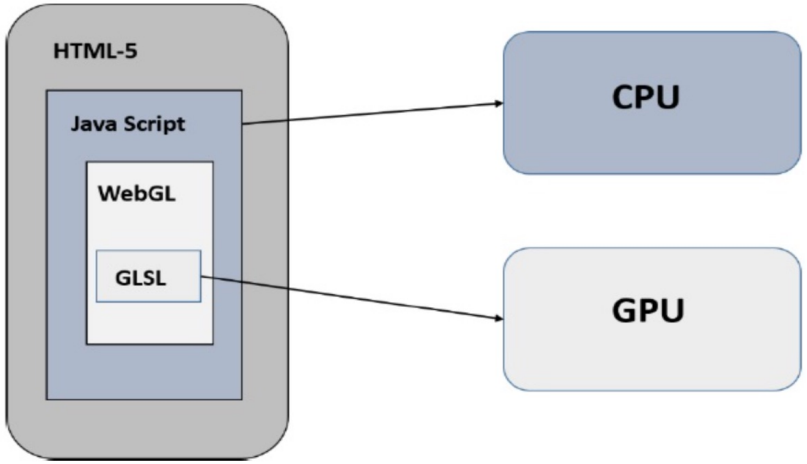
Использовал кадр из видео в качестве текстуры

Спроецировал картинку на воображаемую камеру в центре сферы

Использовал для всего этого WebGL

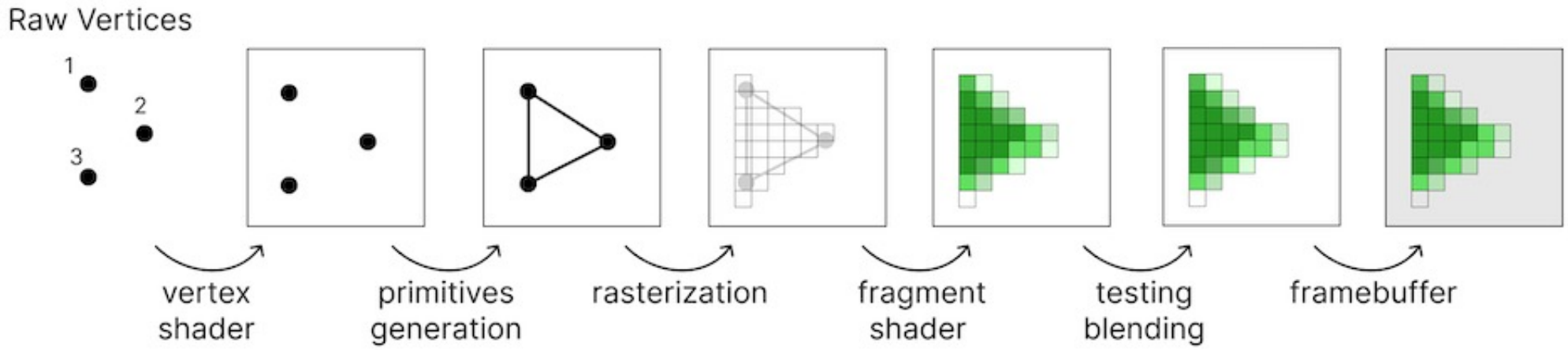
WebGL

WebGL (Web Graphics Library) - программная библиотека для языка JavaScript предназначенная для визуализации интерактивной трёхмерной графики и двухмерной графики в пределах совместимости веб-браузера без использования плагинов. Внутри браузера представлен как элемент <canvas>



GLSL - (OpenGL Shading Language, Graphics Library Shader Language) — си-подобный язык высокого уровня для программирования шейдеров. Основное преимущество – выполнение программы происходит на процессоре видеокарты в то время как javascript-код выполняется на центральном процессоре.

Конвеер WebGL (pipeline)



1 – Загрузка списка вершин
2 – Модификация положения вершин
3 – Создание примитивов
4 – Растеризация
5 – Окраска пикселей
6 – Смешивание
7 - Отрисовка

https://webglfundamentals.org

Создаем и настраиваем canvas

```
/** Создаем канвас */
const canvas = document.createElement( tagName: 'canvas');
canvas.height = 400;
canvas.width = 400;
document.body.appendChild(canvas);

/** Получаем контекст */
const gl = canvas.getContext( contextId: 'webgl');

/** Очищаем канвас */
gl.clearColor( red: 0.5,  green: 0.5,  blue: 0.5,  alpha: 1);
gl.enable(gl.DEPTH_TEST);
gl.clear(gl.COLOR_BUFFER_BIT);
gl.viewport( x: 0,  y: 0, canvas.width, canvas.height);
```

Создаем геометрию и буфер

```
/** Задаем вершины в координатах CLIP SPACE */
const vertices = [-0.5, 0.5, -0.5, -0.5, 0.5, -0.5];

/** Создаем буфер для наших вершин и загружаем данные */
const vertex_buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
gl.bindBuffer(gl.ARRAY_BUFFER,  buffer: null);
```

Создаем вершинный шейдер

```
const vertCode = `
attribute vec2 coordinates;
void main(void) {
    gl_Position = vec4(coordinates, 0.0, 1.0);
}`;
const vertShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertShader, vertCode);
gl.compileShader(vertShader);
```

Создаем фрагментный шейдер

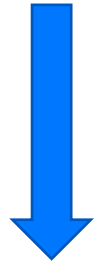
```
const fragCode =`
void main(void) {
    gl_FragColor = vec4(0.0, 0.0, 0.0, 0.1);
}`;
const fragShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragShader, fragCode);
gl.compileShader(fragShader);
```

Создаем шейдерную программу

```
const shaderProgram = gl.createProgram();
gl.attachShader(shaderProgram, vertShader);
gl.attachShader(shaderProgram, fragShader);
gl.linkProgram(shaderProgram);
gl.useProgram(shaderProgram);
```

Прокидываем значения в шейдер

```
const vertCode = `
attribute vec2 coordinates;
void main(void) {
    gl_Position = vec4(coordinates, 0.0, 1.0);
}`;
```



```
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
const coords = gl.getAttribLocation(shaderProgram, name: "coordinates");
gl.vertexAttribPointer(coords, size: 2, gl.FLOAT, normalized: false, stride: 0, offset: 0);
gl.enableVertexAttribArray(coords);
gl.bindBuffer(gl.ARRAY_BUFFER, buffer: null);
```

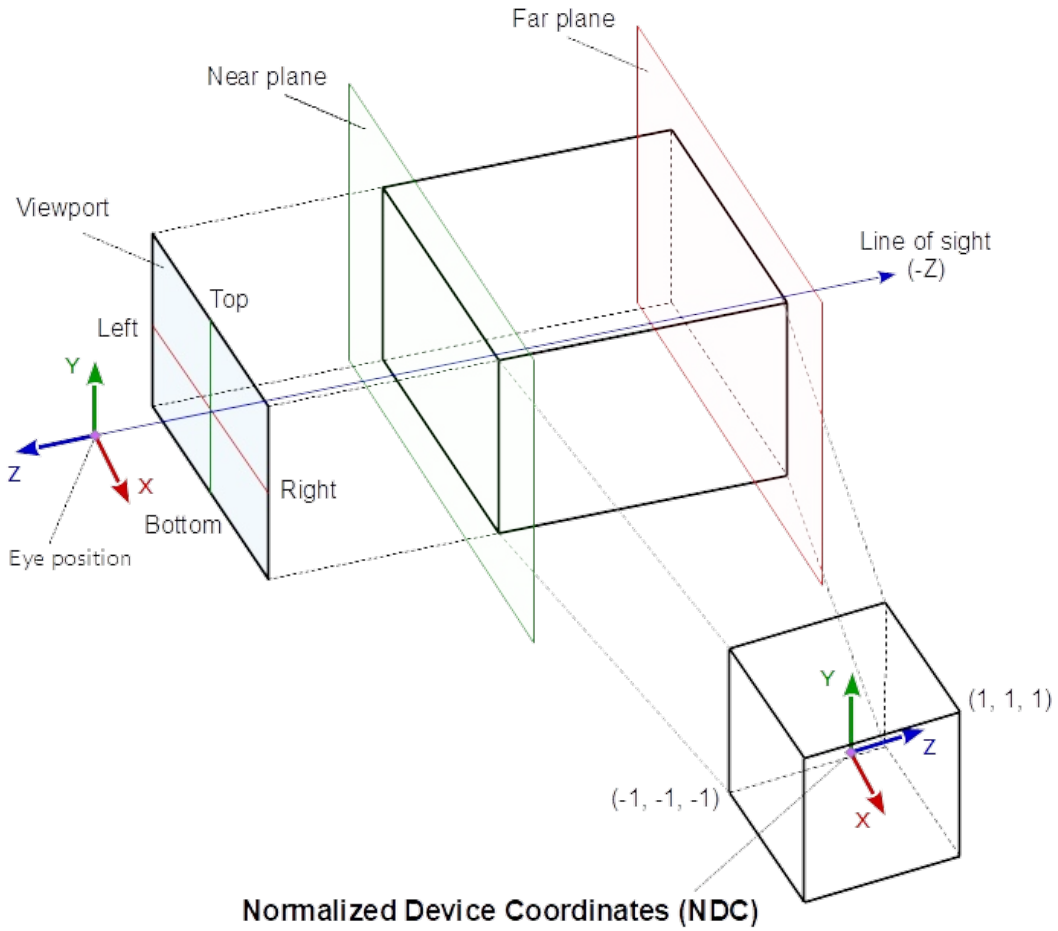
```
/** Задаем вершины в координатах CLIP SPACE */
const vertices = [-0.5, 0.5, -0.5, -0.5, 0.5, -0.5];
```

Рисуем

```
gl.drawArrays(gl.TRIANGLES,  first: 0,  count: 3);
```

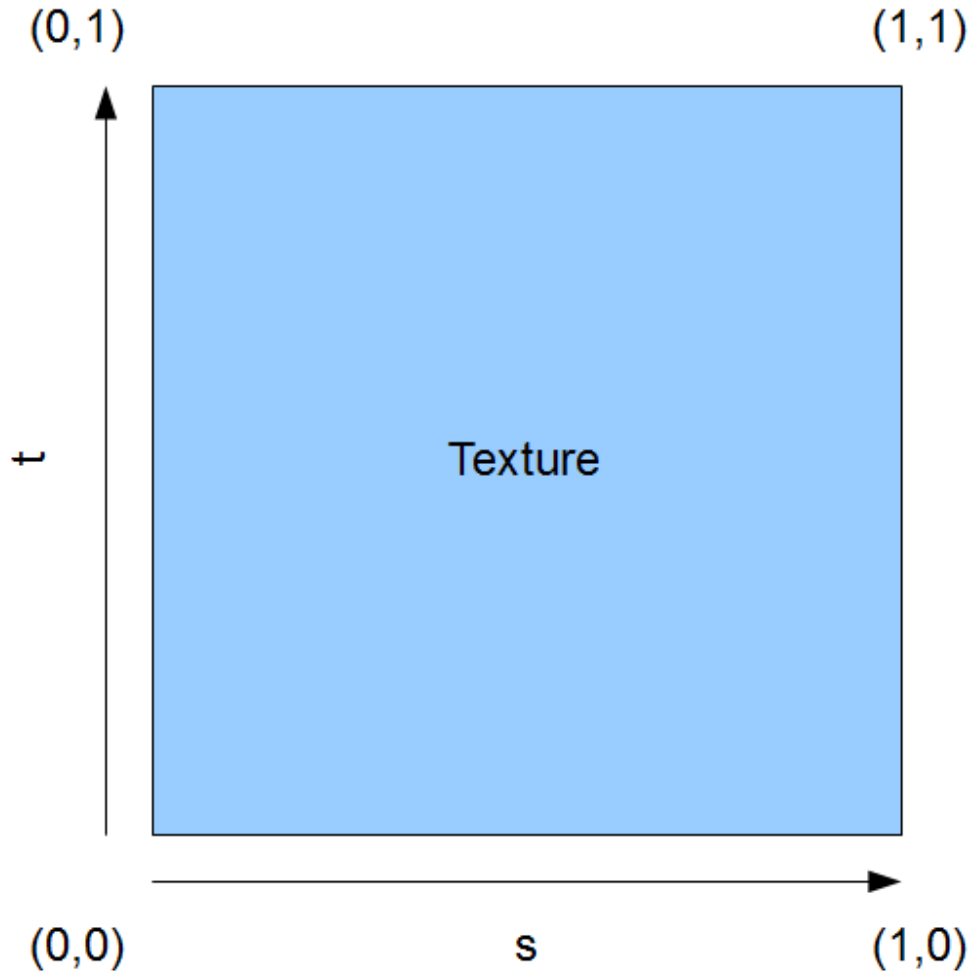
Результат

CLIP SPACE



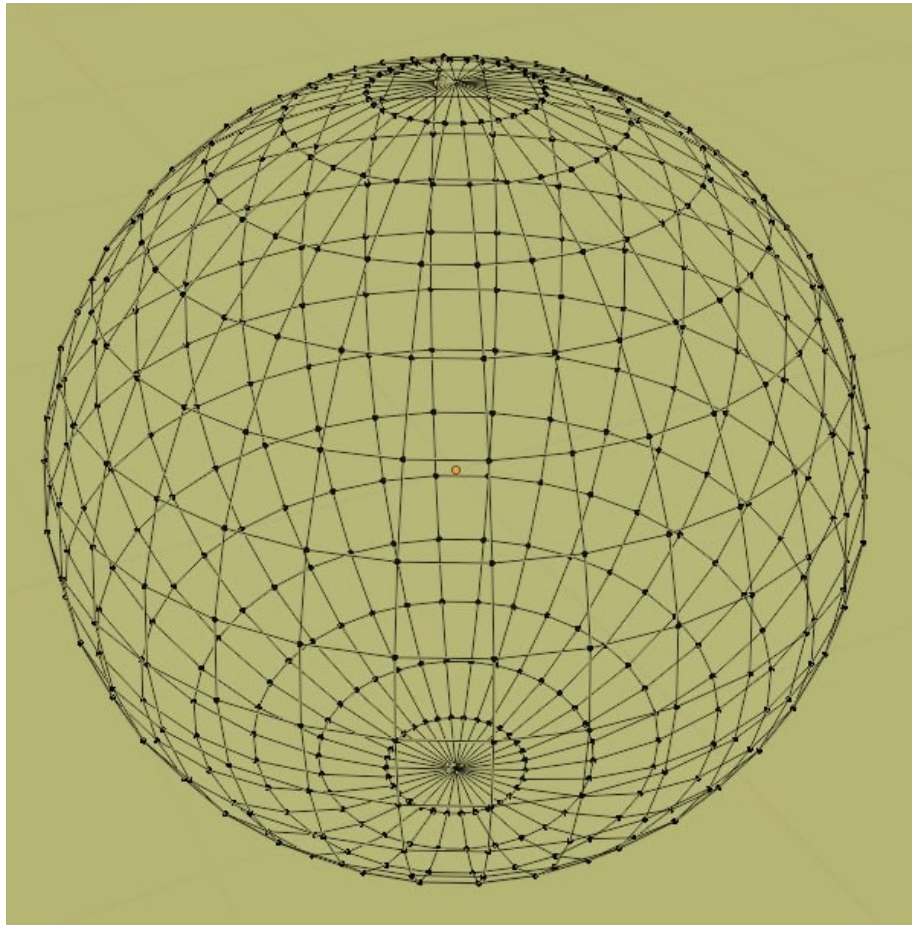
WebGL отрисовывает только ограниченную часть пространства называемую clip space, в котором координаты всегда приводятся к виду [-1..1, -1..1, -1..1].

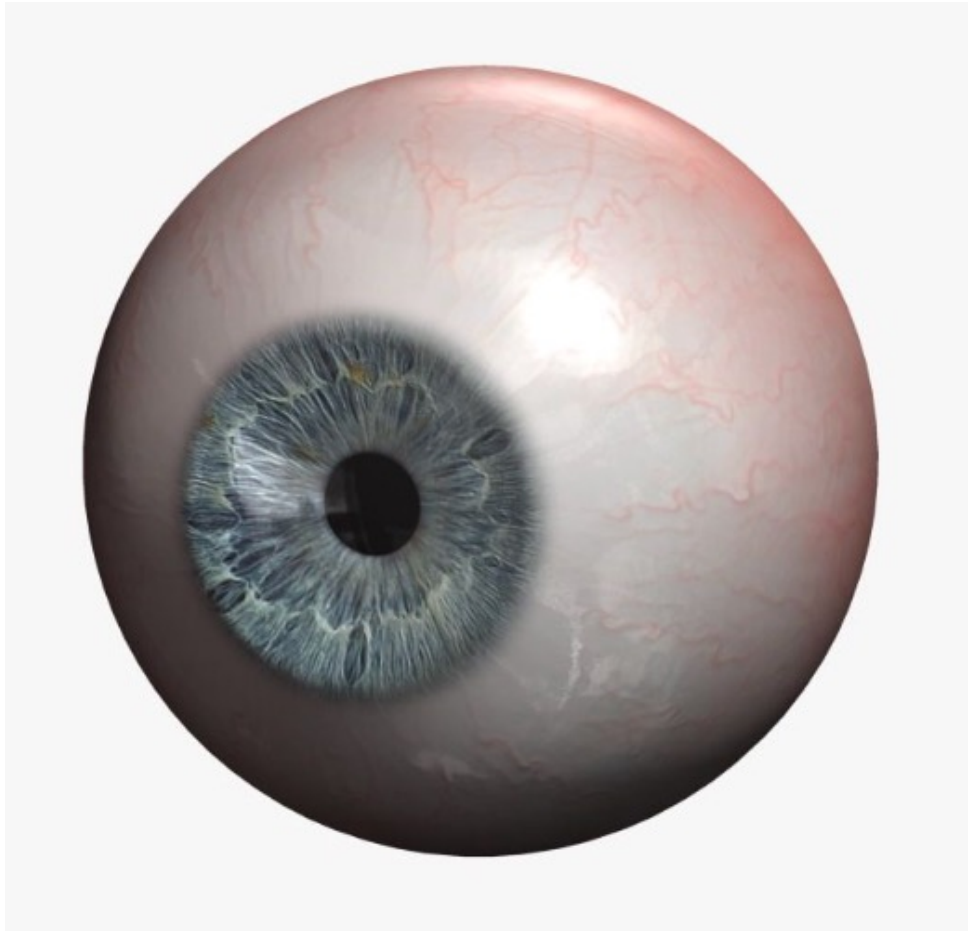
Координаты текстур



Координаты точек текстуры – тексели - всегда отсчитываются от нижнего левого угла и тоже приводятся к нормализованному виду [0..1, 0..1]

Ручное текстурирование сферы





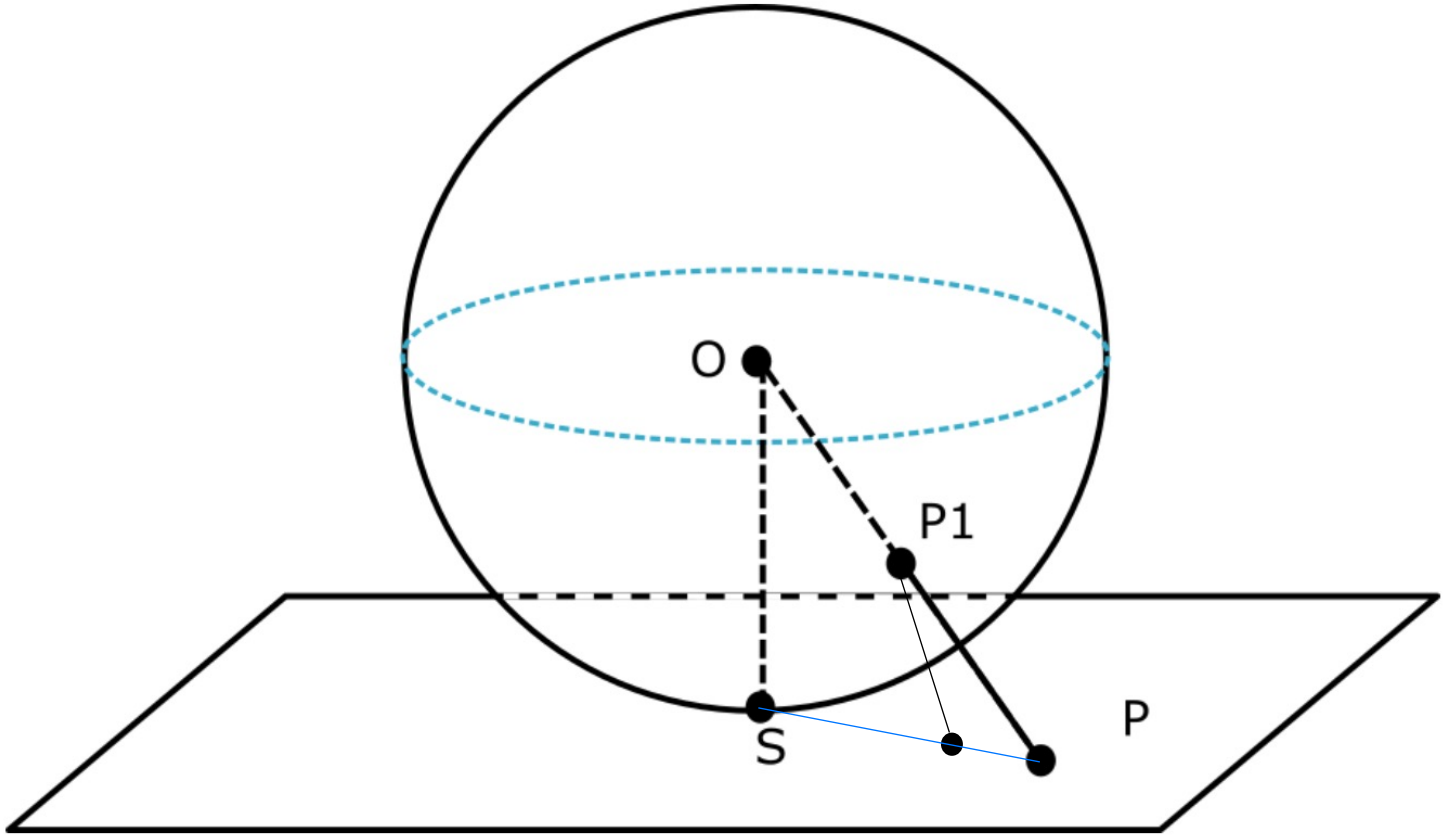


Схема решения

Формула преобразования координат

$$x = \frac{cos\phi\, sin(\lambda - \lambda_0)}{cos(c)}$$

$$y = \frac{cos\phi_1 sin\phi - sin\phi_1 cos\phi cos(\lambda - \lambda_0)}{cos(c)}$$

$$cos(c) = sin\phi_1 sin\phi + cos\phi_1 cos\phi cos(\lambda - \lambda_0)$$

(x, y) – координаты точки на текстуре

(λ_0, ϕ_1)- географические координаты фокуса

(λ, ϕ) – географические координаты текущей точки

Разные типы параметров

```
const vertexLoc = gl.getAttribLocation(shaderProgram, 'a_vertex');
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.vertexAttribPointer(vertexLoc, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vertexLoc);
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

```
const focusLoc = gl.getUniformLocation(shaderProgram, 'u_focus');
gl.uniform2f(focusLoc, u_focus.x, u_focus.y);
```

Создание текстуры

```
videoTexture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, videoTexture);
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
textureLoc = gl.getUniformLocation(shaderProgram, 'u_texture');
```

Обновление текстуры

```
gl.texImage2D(
    gl.TEXTURE_2D,
    0,
    gl.RGBA,
    gl.RGBA,
    gl.UNSIGNED_BYTE,
    sourceVideoElement,
);
```

Маппинг текстуры

```
texelBuffer = gl.createBuffer();
texelLoc = gl.getAttribLocation(shaderProgram, 'a_texel');
gl.bindBuffer(gl.ARRAY_BUFFER, texelBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([0, 0, 1, 0, 1, 1, 0, 1]), gl.STATIC_DRAW);
gl.vertexAttribPointer(texelLoc, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(texelLoc);
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

Вершинный Шейдер (GLSL)

```
attribute vec2 a_vertex;
attribute vec2 a_texel;

varying vec2 v_texel;

void main(void) {
    // direct vertex drawing
    gl_Position = vec4(a_vertex, 0.0, 1.0);
    // save texel vector to pass to fragment shader
    v_texel = a_texel;
}
```

Фрагментный шейдер

```
#ifdef GL_ES
    precision highp float;
    precision highp int;
#else
    precision highp float;
#endif

#define PI 3.14159265358979323846264

varying vec2 v_texel;

uniform sampler2D u_texture;
uniform vec2 u_focus;
```

```
void main(void) {
    float lambda0 = u_focus.x / 360.0;
    float phi0 = u_focus.y / 180.0;

    float lambda = PI * 2.0 * (v_texel.x - 0.5 - lambda0);
    float phi = PI * (v_texel.y - 0.5 - phi0);

    float p = sqrt(lambda * lambda + phi * phi);
    float c = atan(p);
    float cos_c = cos(c);
    float sin_c = sin(c);

    float x = lambda0 + atan(
        lambda * sin_c,
        p * cos(phi0) * cos_c - phi * sin(phi0) * sin_c
    );
    float y = asin(cos_c * sin(phi0) + (phi * sin_c * cos(phi0)) / p);

    vec2 tc = vec2(
        mod(x / (PI * 2.0) - 0.5, 1.0),
        mod(y / PI - 0.5, 1.0)
    );

    gl_FragColor = texture2D(u_texture, tc);
}
```

Результат

Мы справились!

Пример стереоскопического рендеринга

Пример дополненной реальности в WebGL

https://stemkoski.github.io/AR-Examples/

Заключение

Сферическое видео – удобный, современный инструмент презентации пространства. Довольно активно используется в бизнесе для рекламных компаний.

Если немного углубиться в теорию выяснится, что работать с ним достаточно легко.

Надеюсь мой доклад поможет вам в освоении технологии WebGL в целом и сферического видео в частности.

Будет очень круто если вы оставите отзыв и расскажете хотите ли в следующий раз погрузиться в тему поглубже

Будем ВКонтакте!

github.com/verzilka/
videotech-2023

@VERZILKA