Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Лабораторная работа №4

по курсу «Организация графических систем»

Система дополненной реальности

Студенты Попов А.Д.

Группа М-АС-21

Руководитель Кургасов В.В.

доцент

Липецк 2022 г.

Задание

Обработка потокового видеоизображения.

Теоретические сведения

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. WebGL приносит в веб трёхмерную графику, вводя API, который построен на основе OpenGL ES 2.0, что позволяет его использовать в элементах canvas HTML5.

WebGL исполняется как элемент HTML5 и поэтому является полноценной частью объектной модели документа (DOM API) браузера. Может использоваться с любыми языками программирования, которые умеют работать с DOM API, например, JavaScript, Rust, Java и другими. Все ведущие разработчики браузеров Google (Chrome), Mozilla (Firefox), и Apple (Safari), являются членами Khronos и реализуют WebGL в своих браузерах. За счёт использования низкоуровневых средств поддержки OpenGL часть кода на WebGL может выполняться непосредственно на видеокартах. WebGL — это контекст элемента canvas HTML, который обеспечивает API 3D графики без использования плагинов. Первая спецификация была выпущена 3 марта 2011 года. Современная версия 2.0 (несовместима с версией 1.0) доступна с 27 февраля 2017 года.

Поддержка WebGL присутствует в Firefox 4+, Google Chrome 9+, Opera 12+, Safari 5.1+ и Internet Explorer 11+. Однако помимо поддержки WebGL браузером, необходима также его поддержка графическим процессором клиента.

Three.js — легковесная кроссбраузерная библиотека JavaScript, используемая для создания и отображения анимированной компьютерной 3D графики при разработке веб-приложений. Three.js скрипты могут использоваться совместно с элементом HTML5 CANVAS, SVG или WebGL.

Three.js позволяет создавать ускоренную на GPU 3D графику, используя язык JavaScript как часть сайта без подключения проприетарных плагинов для браузера. Это возможно благодаря использованию технологии WebGL.

AR.js — это облегченная библиотека для AR. Она построена на основе библиотеки three.js и jsartoolkit и объединена с A-frame (библиотекой для разработки VR в web) для AR. Тут используются маркеры, чтобы камера могла обнаруживать и отображать AR объект. В этом методе создания дополненной реальности замечательно то, что он:

Реализация системы

Листинг 1 – index.html

```
<!DOCTYPE html>
<head>
   <meta name="viewport" content="width=device-width, user-scalable=no,
minimum-scale=1.0, maximum-scale=1.0">
   <title>Globe</title>
   <script src='js/keyboard.js'></script>
   <script src='js/three.js'></script>
   <script src="jsartoolkit5/artoolkit.min.js"></script>
   <script src="jsartoolkit5/artoolkit.api.js"></script>
   <script src="threex/threex-artoolkitsource.js"></script>
   <script src="threex/threex-artoolkitcontext.js"></script>
   <script src="threex/threex-arbasecontrols.js"></script>
   <script src="threex/threex-armarkercontrols.js"></script>
</head>

<body style='margin : 0px; overflow: hidden; font-family: Monospace;'>

<script>

var scene, camera, renderer, clock, deltaTime, totalTime, keyboard;

var arToolkitSource, arToolkitContext;

var markerNames, markerArray, currentMarkerName;

var sceneGroup;

var globe;

initialize();
animate();

function initialize()
{
   scene = new THREE.Scene();

   let ambientLight = new THREE.AmbientLight( 0xcccccc, 0.5 );
   scene.add( ambientLight );

   camera = new THREE.Camera();
   scene.add(camera);

   renderer = new THREE.WebGLRenderer({
      antialias : true,
      alpha: true
   });
   renderer.setClearColor(new THREE.Color('lightgrey'), 0)
   renderer.setSize( 800, 600 );
   renderer.domElement.style.position = 'absolute'
```

```
    renderer.domElement.style.top = '0px'
    renderer.domElement.style.left = '0px'
    document.body.appendChild( renderer.domElement );

    clock = new THREE.Clock();
    deltaTime = 0;
    totalTime = 0;
    keyboard = new Keyboard();

    ////////////////////////////////////////////////////////////
    // setup arToolkitSource
    ////////////////////////////////////////////////////////////

    arToolkitSource = new THREEx.ArToolkitSource({
        sourceType : 'webcam',
    });

    function onResize()
    {
        arToolkitSource.onResize()
        arToolkitSource.copySizeTo(renderer.domElement)
        if ( arToolkitContext.arController !== null )
        {
            arToolkitSource.copySizeTo(arToolkitContext.arController.canvas)
        }
    }

    arToolkitSource.init(function onReady(){
        onResize()
    });

    // handle resize event
    window.addEventListener('resize', function(){
        onResize()
    });

    ////////////////////////////////////////////////////////////
    // setup arToolkitContext
    ////////////////////////////////////////////////////////////

    // create atToolkitContext
    arToolkitContext = new THREEx.ArToolkitContext({
        cameraParametersUrl: 'data/camera_para.dat',
        detectionMode: 'mono',
        maxDetectionRate: 30,
    });

    // copy projection matrix to camera when initialization complete
    arToolkitContext.init( function onCompleted(){
        camera.projectionMatrix.copy( arToolkitContext.getProjectionMatrix()
);
    });

    ////////////////////////////////////////////////////////////
    // setup markerRoots
    ////////////////////////////////////////////////////////////

    markerNames = ["kanji", "letterA", "letterB"];

    markerArray = [];

    for (let i = 0; i < markerNames.length; i++)
    {
        let marker = new THREE.Group();
```

```
      scene.add(marker);
      markerArray.push(marker);

      let markerControls = new THREEx.ArMarkerControls(arToolkitContext,
marker, {
         type: 'pattern', patternUrl: "data/" + markerNames[i] + ".patt",
      });

      let markerGroup = new THREE.Group();
      marker.add(markerGroup);
   }

   ////////////////////////////////////////////////////////////
   // setup scene
   ////////////////////////////////////////////////////////////

   sceneGroup = new THREE.Group();

   let loader = new THREE.TextureLoader();

   let geometry1 = new THREE.SphereGeometry(1, 32,32);
   let texture = loader.load( 'images/earth-sphere.jpg' );
   let material1 = new THREE.MeshLambertMaterial( { map: texture, opacity:
0.75 } );
   globe = new THREE.Mesh( geometry1, material1 );
   globe.position.y = 1;
   sceneGroup.add(globe);

   markerArray[0].children[0].add( sceneGroup );
   currentMarkerName = markerNames[0];

   let pointLight = new THREE.PointLight( 0xffffff, 1, 50 );
   camera.add( pointLight );
}


function update()
{
   keyboard.update();

   globe.rotation.y += 0.01;

   let anyMarkerVisible = false;
   for (let i = 0; i < markerArray.length; i++)
   {
      if ( markerArray[i].visible )
      {
         anyMarkerVisible = true;
         markerArray[i].children[0].add( sceneGroup );
         if ( currentMarkerName != markerNames[i] )
         {
            currentMarkerName = markerNames[i];
            // console.log("Switching to " + currentMarkerName);
         }

         let p = markerArray[i].children[0].getWorldPosition();
         let q = markerArray[i].children[0].getWorldQuaternion();
         let s = markerArray[i].children[0].getWorldScale();
         let lerpAmount = 0.5;

         scene.add(sceneGroup);
         sceneGroup.position.lerp(p, lerpAmount);
         sceneGroup.quaternion.slerp(q, lerpAmount);
         sceneGroup.scale.lerp(s, lerpAmount);
```

```
            break;
        }
    }

    if ( !anyMarkerVisible )
    {
        // console.log("No marker currently visible.");
    }

    let baseMarker = markerArray[0];

    // update relative positions of markers
    for (let i = 1; i < markerArray.length; i++)
    {
        let currentMarker = markerArray[i];
        let currentGroup  = currentMarker.children[0];
        if ( baseMarker.visible && currentMarker.visible )
        {
            // console.log("updating marker " + i " -> base offset");

            let relativePosition = currentMarker.worldToLocal(
baseMarker.position.clone() );
            currentGroup.position.copy( relativePosition );

            let relativeRotation =
currentMarker.quaternion.clone().inverse().multiply(
baseMarker.quaternion.clone() );
            currentGroup.quaternion.copy( relativeRotation );
        }
    }

    // update artoolkit on every frame
    if ( arToolkitSource.ready !== false )
        arToolkitContext.update( arToolkitSource.domElement );

}


function render()
{
    renderer.render( scene, camera );
}


function animate()
{
    requestAnimationFrame(animate);
    deltaTime = clock.getDelta();
    totalTime += deltaTime;
    update();
    render();
}

</script>

</body>
</html>
```

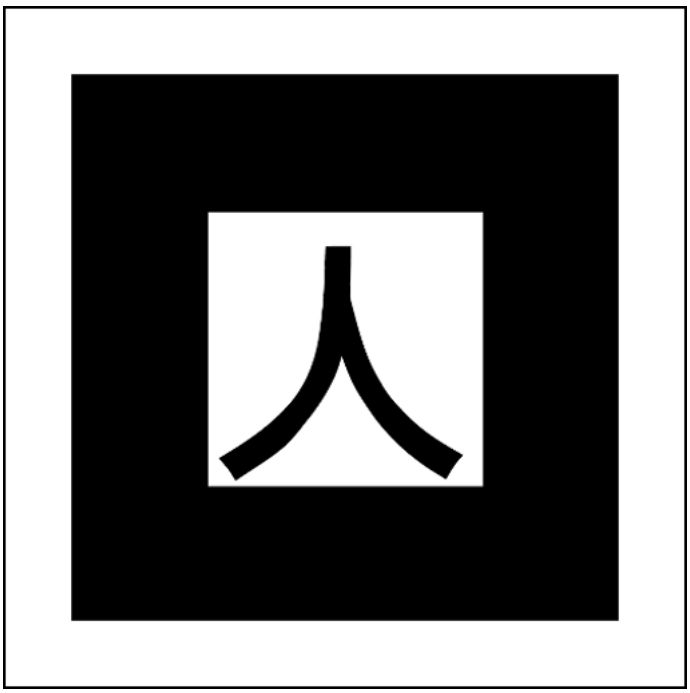

Рисунок 1 - Маркер

Результат работы

Рисунок 2 - Работа на пк

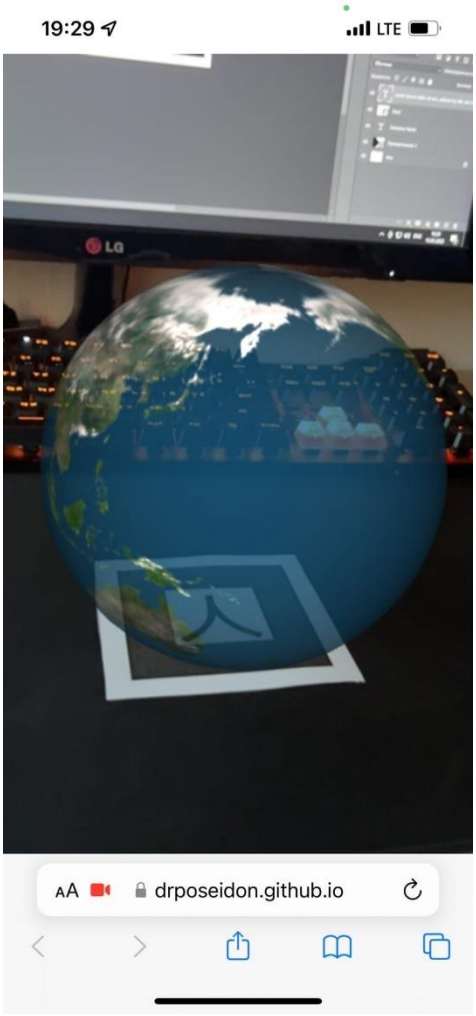

Рисунок 3 - Работа на смартфоне

Вывод

В результате лабораторной работы была разработана система в виде веб-приложения которая позволяет просмотреть в режиме дополненной реальности 3D модель земли.

Ссылка на приложение: https://drposeidon.github.io/lab4-arjs/

Ссылка на github: https://github.com/DrPoseidon/lab4-arjs