Universitetet i Agder
Fakultet for fakultet for Teknologi og realfag

# E K S A M E N

| | |
|---|---|
| **Emnekode:** | **DAT200** |
| **Emnenavn:** | **Grafisk Databehandling** |
| Dato: | 23. November 2016 |
| Varighet: | 0900 - 1300 |
| Antall sider inkl. forside | 8 |
| Tillatte hjelpemidler: | Skrivesaker |
| Merknader: | Oppgavenes vekting er angitt i overskrift på hver oppgave. |

**OPPGAVE 1. (12%)**

NB: Du får + 1 poeng for riktig svar, - ½ poeng for feil svar.
(Skriv besvarelsen inn på eget ark sammen med resten av besvarelsen.)
Angi om du er enig(Ja) eller uenig(Nei) i følgende utsagn:

| | | Ja | Nei |
|---|---|---|---|
| a) | En B-Spline interpolerer sine kontrollpunkt og approksimerer sine knutepunkter. | | |
| b) | Ved bruk av lyspenn så fikk man mulighet for å angi posisjoner på datamaskinen. | | |
| c) | Z-buffer algoritmen (Dybdebufferalgoritmen) er en objektpresisjons-algoritme («objekt precicion») for fjerning av skjulte flater. | | |
| d) | Fordelen med $C^1$ kontinuitet i forhold til $G^1$ kontinuitet er at kurven blir glattere og får bedre egenskaper i forhold til f. eks luft- og vannmotstand. | | |
| e) | En perspektivisk projeksjon fører til at fjerne ting blir forholdsvis større enn nære ting. | | |
| f) | Når man har en "Binary Space Partitioning Tree"-representasjon av en solid, er det en enkel sak å gjennomføre fjerning av skjulte linjer og flater fra alle posisjoner i rommet. | | |
| g) | Dersom høyde/bredde forholdet er ulikt i overføringen fra et vindu til en skjermport («ViewPort») risikerer vi at sirkler blir ellipser når vi ser modellen på en dataskjerm. | | |
| h) | En isometrisk projeksjon har et projeksjonsplan som skjærer aksene i lik avstand fra origo. | | |
| i) | Ved bruk av "antialiasing" oppnås en raskere gjengivelse av linjer på en dataskjerm. | | |
| j) | Sutherland Hodgman sin klippingsalgoritme kan kun anvendes for kvadratiske klippeområder. | | |
| k) | Et matt materiale har specular refleksjon over at større område enn et blankt materiale. | | |
| l) | Vi bruker homogene koordinater fordi det letter arbeidet ved uttegning av sirkler. | | |

## OPPGAVE 2. TRANSFORMASJONER (20%)

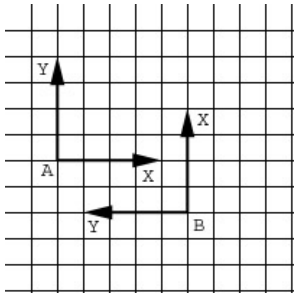
For hver underoppgave a)-d) skriv ned en matrise med homogene koordinater som gjør nevnte transformasjon. Vis matrisene for både 2D- og 3D-transformasjoner der det gir mening.

a) Lag et objekt fire ganger så stort

b) Reduser x-retningen med 50% og øk y-retningen med 50%.

c) Roter et objekt 90 grader om x-aksen.

d) Vi har to koordinatsystem A og B:




Sett opp de transformasjonene som er nødvendige for å overføre geometri fra koordinatsystem B til koordinatsystem A.

e) Når vi arbeider med rotasjoner og skifte av koordinatsystem så kan vi av og til utnytte egenskapen at en matrise er spesiell ortogonal. Hva betyr det og hvordan kan dette anvendes?

## OPPGAVE 3. FARGER, SKULTE FLATER, SOLIDER OG INTENSITETER (20%)

a) Hvorfor må vi anvende to forskjellige fargemodeller når farger skal vises på papir og på skjerm?

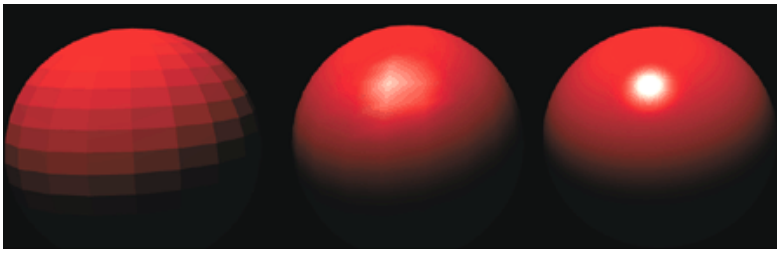
b) Hva heter den mest brukte algoritmen for fjerning av skjulte linjer og flater?

c) Forklar virkemåten til algoritmen i b) og angi om den har noen svakheter.

d) Hva forstår vi med en Octree-representasjon av et 3D objekt?

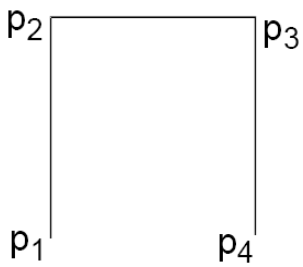
e) I figuren nedenfor er det vist en kule med 3 forskjellige algoritmer for intensitetsbestemmelse. De tre algoritmene er konstant intensitet, Gouraud Shading og Phong Shading. Hvilken algoritme hører til hvilket bilde?

A: B: C:



## OPPGAVE 4. KURVER, KLIPPING OG PROJEKSJONER (24%)

Gitt følgende kontrollpolygon:




a) Tegn av kontrollpolygonet og skisser en kubisk Bèzier-kurve i polygonet

b) Tegn av kontrollpolygonet og skisser en kubisk uniform B-spline-kurve i polygonet.

c) Nevn de vesentligste forskjellene mellom kubiske Bèzierkurver og uniforme kubiske B-splineskurver.

d) Hvordan kan vi sikre oss at vi får en sammenhengende kontinuerlig (G1 kontinuitet) Bèzier-kurve når vi skal knytte sammen to Bèzier-kurvesegmenter?

e) Forklar grunnprinsippet for Sutherland-Hodgman sin polygonklippingsalgoritme.

f) Regn ut projeksjonen av punktet (20,30,40) ned i x-y-planet når projeksjonspunktet er i (0,0,-50).

## OPPGAVE 5. 3D-Studio (10%)

Vi skal modellere deler av et satellittanlegg til bobil (Se figurer nedenfor).

a) Forklar kort og prinsipielt hvordan du vil gå frem i 3D-Studio for å modellere selve parabolen til satellittanlegget samt de 4 boltehodene som du kan se i parabolflaten. Ta dine egne antagelser dersom du er usikker på formen på objektene.

b) Hvilke materiale vil du velge i 3D Studio Max for henholdsvis parabolen og boltene og forklar hvordan du vil få frem teksten og logoen på parabolen?

## OPPGAVE 6. Three.js (14%)

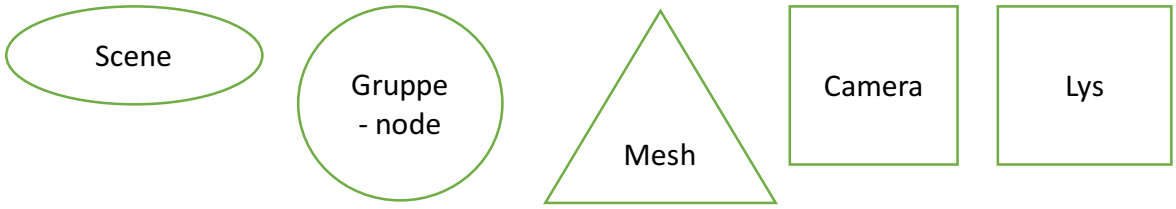
Vi skal nå laste inn og animere (deloppgave b nedenfor) parabolanlegget fra oppgave 5 i Three.js. Den består av følgende 4 deler:

- Parabol (Selve parabolen som reflekterer signalene samt dens innfestning til fundamentet),
- Hodet (Den enheten som tar imot signalene. Inkluderer også armen som hodet er festet i)
- Fundament (Den delen som hodet og parabolen er koblet til)
- Plate (Den delen som fundamentet er forbundet til som er festet fast til bobilen).

Når parabolen brukes kan både Parabol, Hodet og Fundamentet rotere slik at rett posisjon kan innstilles i forhold til satellitten som man ønsker å finne.

a) Når man skal eksportere objekter fra 3D Studio Max til Three.js er det en del ting man bør passe på å gjøre før man eksporterer objektene. Kan du angi noen viktige ting som man bør gjøre på forhånd før man gjennomfører eksporten?

b) Vi vil nå ha muligheten for å bevege parabolen slik at den kan stilles inn i en posisjon der den kan ta imot signalene fra en satellitt. Tegn i diagrams-form det treet, med nødvendige forklaringer, som skal til for å danne et bevegelig parabolanlegg med lyssetting og kamera. Bruk følgende elementer i scene-grafen (treet) når denne skal tegnes:



## Vedlegg 1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Pyramide</title>
    <script type="text/javascript" src="three.js"></script>
    <script type="text/javascript" src="stats.js"></script>
    <script type="text/javascript" src="dat.gui.js"></script>
    <script type="text/javascript" src="ColladaLoader.js"></script>

    <style>
        body {
            /* set margin to 0 and overflow to hidden, to go fullscreen */
            margin: 0;
            overflow: hidden;
        }
    </style>
</head>
<body>
<div id="Stats-output">
</div>
<!-- Div which will hold the Output -->
<div id="WebGL-output">
</div>

<!-- Javascript code -->
<script type="text/javascript">

    // once everything is loaded, we run our Three.js stuff.
    function init() {

        var stats = initStats();

        // create a scene, that will hold all our elements such as objects,
        // cameras and lights.
        var scene = new THREE.Scene();

        var branchGroup = new THREE.Object3D();

        // create a camera, which defines where we're looking at.
        var camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight,
        0.1, 1000);

        // create a render and set the size
        var renderer = new THREE.WebGLRenderer();
        renderer.setClearColor(new THREE.Color(0xEEEEEE, 1.0));
        renderer.setSize(window.innerWidth, window.innerHeight);
        renderer.shadowMapEnabled = true;

        // create helper axes
        var axes = new THREE.AxisHelper(5);
        scene.add(axes);

        // create the ground plane
        var planeGeometry = new THREE.PlaneGeometry(20, 20);
        var planeMaterial = new THREE.MeshPhongMaterial({color: 0xffffff});
        var plane = new THREE.Mesh(planeGeometry, planeMaterial);
        plane.receiveShadow = true;

        // rotate and position the plane
```

```
plane.rotation.x = -0.5 * Math.PI;
plane.position.x = 0;
plane.position.y = -5;
plane.position.z = 0;

// add the plane to the scene
scene.add(plane);

var blad1 = createObject("Blad.dae", "test");
var blad2 = createObject("Blad.dae", "blad2");
var blad3 = createObject("Blad.dae", "blad3");
var fundament = createObject("Fundament.dae", "fundament");
branchGroup.add(fundament);

// Connect the blades and add the Windmill to the scene
blad3.rotateZ(2 * Math.PI / 3);
blad2.rotateZ(2 * Math.PI / 3);
blad1.add(blad2);
blad2.add(blad3);
branchGroup.add(blad1);
scene.add(branchGroup);

// position and point the camera to the center of the scene
camera.position.x = 17;
camera.position.y = 17;
camera.position.z = 17;
camera.lookAt(scene.position);

// add subtle ambient lighting
var ambientLight = new THREE.AmbientLight();
//var ambientLight = new THREE.AmbientLight(color,0.1);
//ambientLight.color.setRGB( r * intensity, g * intensity, b * intensity );
var intensity = 0.2;
ambientLight.color.setRGB(1 * intensity, 1 * intensity, 1 * intensity);
scene.add(ambientLight);

// add spotlight for the shadows
var spotLight = new THREE.SpotLight();
spotLight.color.setRGB(1, 1, 1);
spotLight.position.set(20, 80, 20);
spotLight.castShadow = true;
spotLight.angle = Math.PI / 50;
spotLight.shadowMapWidth = 1024;
spotLight.shadowMapHeight = 1024;
//spotLight.shadowMapSize=[1024,1024];
spotLight.lookAt(scene);
scene.add(spotLight);

// add the output of the renderer to the html element
document.getElementById("WebGL-output").appendChild(renderer.domElement);

// call the render function
var step = 0;
var controls = new function () {
    this.rotationSpeed = 0.02;
    this.zoom = 7;
    this.angle = 0;
};
var gui = new dat.GUI();
gui.add(controls, 'rotationSpeed', 0, 0.5);
gui.add(controls, 'zoom', 2, 100);
gui.add(controls, 'angle', 0, 360);
var angle = 0;
```

```
        var radius = 7;
        render();

        function createObject(objFile, objName) {
            var container = new THREE.Object3D();
            var loader = new THREE.ColladaLoader();
            var mesh = new THREE.Mesh;
            loader.load(objFile, function (result) {
                mesh = result.scene.children[0].children[0].clone();
                mesh.scale.set(0.1, 0.1, 0.1);
                mesh.castShadow = true;
                mesh.name = objName;
                container.add(mesh);
            });
            return container;
        }


        function render() {
            stats.update();
            // rotate the Windmill blades around its center
            blad1.rotation.z += controls.rotationSpeed;

            angle = controls.angle * Math.PI / 180;
            radius = controls.zoom;
            camera.position.x = radius * Math.cos(angle);
            camera.position.z = radius * Math.sin(angle);
            camera.position.y = radius;
            camera.lookAt(scene.position);

            // render using requestAnimationFrame
            requestAnimationFrame(render);
            renderer.render(scene, camera);
        }

        function initStats() {

            var stats = new Stats();
            stats.setMode(0); // 0: fps, 1: ms
            // Align top-left
            stats.domElement.style.position = 'absolute';
            stats.domElement.style.left = '0px';
            stats.domElement.style.top = '0px';
            document.getElementById("Stats-output").appendChild(stats.domElement);
            return stats;
        }
    }
    window.onload = init;
</script>
</body>
</html>
```