**SPRAWOZDANIE**

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk

# **Laboratorium**

Data: 11.03.2024

**Temat:** "Modelowanie hierarchiczne w grafice 2D"

# **Wariant:**

# Jedenastokąt

Mateusz Żelazo

Informatyka I stopnia

stacjonarne, 4 semestr

Gr.3a

1. **Polecenie:**

Opracować scenę hierarchiczną zgodnie z obrazem używając zamiast kół wielokąty obracające się (animacja!) według wariantu. Opracowanie powinno być w jednym z języków: Java lub JavaScript, na dwa sposoby:

(a) używając hierarchiję funkcje (sposób subroutinowy)

(b) tworząc graf sceny (sposób obiektowy). W tym celu proponuję do pobrania odpowiedni pliki

**Podpunkt a (sposób subroutinowy):**

1. **Wprowadzane dane:**
   1. **Rysowanie trójkątów**

**let px = 0;**

**let py = 0;**

**let scale = 1;**

**graphics.save()**

**graphics.fillStyle = "blue";**

**graphics.translate(1.5 \* scale + px, -0.25 \* scale + py)**

**graphics.scale(scale, scale);**

**graphics.rotate((frameNumber \* 0.75) \* Math.PI / 180);**

**filledPolygon();**

**graphics.restore();**

**graphics.save()**

**graphics.fillStyle = "blue";**

**graphics.translate(-1.5 \* scale + px, 0.25 \* scale + py)**

**graphics.scale(scale, scale);**

**graphics.rotate((frameNumber \* 0.75) \* Math.PI / 180);**

**filledPolygon();**

**graphics.restore();**

**graphics.save()**

**graphics.fillStyle = "red";**

**graphics.rotate(-10 / 180 \* Math.PI)**

**graphics.scale(3 \* scale, 0.2 \* scale)**

**graphics.translate(px, py);**

**filledRect();**

**graphics.restore();**

**graphics.save()**

**graphics.scale(0.5 \* scale, 2 \* scale)**

**graphics.fillStyle = "blue";**

**graphics.translate(0 \* scale + px, -1 \* scale + py)**

**filledTriangle();**

**graphics.restore();**

***// --------------------- 2***

**px = -2;**

**py = 2;**

**scale = 0.5;**

**graphics.save()**

**graphics.fillStyle = "blue";**

**graphics.translate(1.5 \* scale + px, -0.25 \* scale + py)**

**graphics.scale(scale, scale);**

**graphics.rotate((frameNumber \* 0.75) \* Math.PI / 180);**

**filledPolygon();**

**graphics.restore();**

**graphics.save()**

**graphics.fillStyle = "blue";**

**graphics.translate(-1.5 \* scale + px, 0.25 \* scale + py)**

**graphics.scale(scale, scale);**

**graphics.rotate((frameNumber \* 0.75) \* Math.PI / 180);**

**filledPolygon();**

**graphics.restore();**

**graphics.save()**

**graphics.fillStyle = "red";**

**graphics.translate(px, py);**

**graphics.rotate(-10 / 180 \* Math.PI)**

**graphics.scale(3 \* scale, 0.2 \* scale)**

**filledRect();**

**graphics.restore();**

**graphics.save()**

**graphics.translate(0 \* scale + px, -2 \* scale + py)**

**graphics.scale(0.5 \* scale, 2 \* scale)**

**graphics.fillStyle = "blue";**

**filledTriangle();**

**graphics.restore();**

***// --------------------- 3***

**px = 2;**

**py = 2;**

**scale = 0.3;**

**graphics.save()**

**graphics.fillStyle = "blue";**

**graphics.translate(1.5 \* scale + px, -0.25 \* scale + py)**

**graphics.scale(scale, scale);**

**graphics.rotate((frameNumber \* 0.75) \* Math.PI / 180);**

**filledPolygon();**

**graphics.restore();**

**graphics.save()**

**graphics.fillStyle = "blue";**

**graphics.translate(-1.5 \* scale + px, 0.25 \* scale + py)**

**graphics.scale(scale, scale);**

**graphics.rotate((frameNumber \* 0.75) \* Math.PI / 180);**

**filledPolygon();**

**graphics.restore();**

**graphics.save()**

**graphics.translate(px, py);**

**graphics.fillStyle = "red";**

**graphics.rotate(-10 / 180 \* Math.PI)**

**graphics.scale(3 \* scale, 0.2 \* scale)**

**filledRect();**

**graphics.restore();**

**graphics.save()**

**graphics.translate(0 \* scale + px, -2 \* scale + py)**

**graphics.scale(0.5 \* scale, 2 \* scale)**

**graphics.fillStyle = "blue";**

**filledTriangle();**

**graphics.restore();**

* 1. **Rysowanie jedenastokąta**

**function filledPolygon() {**

**graphics.beginPath();**

**let w = 11;**

**let r = 0.5;**

**let kat = 360 / w;**

**for (let i = 1; i <= w; i++) {**

**let rad = ((kat \* i) \* (2 \* Math.PI)) / 360**

**let y = Math.sin(rad) \* r;**

**let x = Math.cos(rad) \* r;**

**graphics.lineTo(x, y);**

**}**

**graphics.closePath();**

**graphics.fill();**

**}**

1. **Wykorzystane komendy:**
   1. **kod źródłowy**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="UTF-8">**

**<script>**

**var canvas; *// The canvas that is used as the drawing surface***

**var graphics; *// The 2D graphics context for drawing on the canvas.***

**var X\_LEFT = -4; *// The xy limits for the coordinate system.***

**var X\_RIGHT = 4;**

**var Y\_BOTTOM = -3;**

**var Y\_TOP = 3;**

**var BACKGROUND = "white"; *// The display is filled with this color before the scene is drawn.***

**var pixelSize; *// The size of one pixel, in the transformed coordinates.***

**var frameNumber = 0; *// Current frame number. goes up by one in each frame.***

***// TODO: Define any other necessary state variables.***

***/\*\****

***\* Responsible for drawing the entire scene. The display is filled with the background***

***\* color before this function is called.***

***\*/***

**function drawWorld() {**

***// TODO: Draw the content of the scene.***

**rotatingRect(); *// (DELETE THIS EXAMPLE)***

**}**

***/\*\****

***\* This method is called just before each frame is drawn. It updates the modeling***

***\* transformations of the objects in the scene that are animated.***

***\*/***

**function updateFrame() {**

**frameNumber++;**

***// TODO: If other updates are needed for the next frame, do them here.***

**}**

***// TODO: Define methods for drawing the objects in the scene.***

**function rotatingRect() { *// (DELETE THIS EXAMPLE)***

**graphics.save(); *// (It might be necessary to save/restore transform and color)***

**graphics.fillStyle = "red";**

**graphics.rotate( (frameNumber\*0.75) \* Math.PI/180 );**

**graphics.scale( 2, 2 );**

**filledRect();**

**graphics.restore();**

**}**

***//------------------- Some methods for drawing basic shapes. ----------------***

**function line() { *// Draws a line from (-0.5,0) to (0.5,0)***

**graphics.beginPath();**

**graphics.moveTo(-0.5,0);**

**graphics.lineTo(0.5,0);**

**graphics.stroke();**

**}**

**function rect() { *// Strokes a square, size = 1, center = (0,0)***

**graphics.strokeRect(-0.5,-0.5,1,1);**

**}**

**function filledRect() { *// Fills a square, size = 1, center = (0,0)***

**graphics.fillRect(-0.5,-0.5,1,1);**

**}**

**function circle() { *// Strokes a circle, diameter = 1, center = (0,0)***

**graphics.beginPath();**

**graphics.arc(0,0,0.5,0,2\*Math.PI);**

**graphics.stroke();**

**}**

**function filledCircle() { *// Fills a circle, diameter = 1, center = (0,0)***

**graphics.beginPath();**

**graphics.arc(0,0,0.5,0,2\*Math.PI);**

**graphics.fill();**

**}**

**function filledTriangle(g2) {*// Filled Triangle, width 1, height 1, center of base at (0,0)***

**g2.beginPath();**

**g2.moveTo(-0.5,0);**

**g2.lineTo(0.5,0);**

**g2.lineTo(0,1);**

**g2.closePath();**

**g2.fill();**

**}**

***// ------------------------------- graphics support functions --------------------------***

***/\*\****

***\* Draw one frame of the animation. Probably doesn't need to be changed,***

***\* except maybe to change the setting of preserveAspect in applyLimits().***

***\*/***

**function draw() {**

**graphics.save(); *// to make sure changes don't carry over from one call to the next***

**graphics.fillStyle = BACKGROUND; *// background color***

**graphics.fillRect(0,0,canvas.width,canvas.height);**

**graphics.fillStyle = "black";**

**applyLimits(graphics,X\_LEFT,X\_RIGHT,Y\_TOP,Y\_BOTTOM,false);**

**graphics.lineWidth = pixelSize; *// Use 1 pixel as the default line width***

**drawWorld();**

**graphics.restore();**

**}**

***/\*\****

***\* Applies a coordinate transformation to the graphics context, to map***

***\* xleft,xright,ytop,ybottom to the edges of the canvas. This is called***

***\* by draw(). This does not need to be changed.***

***\*/***

**function applyLimits(g, xleft, xright, ytop, ybottom, preserveAspect) {**

**var width = canvas.width; *// The width of this drawing area, in pixels.***

**var height = canvas.height; *// The height of this drawing area, in pixels.***

**if (preserveAspect) {**

***// Adjust the limits to match the aspect ratio of the drawing area.***

**var displayAspect = Math.abs(height / width);**

**var requestedAspect = Math.abs(( ybottom-ytop ) / ( xright-xleft ));**

**var excess;**

**if (displayAspect > requestedAspect) {**

**excess = (ybottom-ytop) \* (displayAspect/requestedAspect - 1);**

**ybottom += excess/2;**

**ytop -= excess/2;**

**}**

**else if (displayAspect < requestedAspect) {**

**excess = (xright-xleft) \* (requestedAspect/displayAspect - 1);**

**xright += excess/2;**

**xleft -= excess/2;**

**}**

**}**

**var pixelWidth = Math.abs(( xright - xleft ) / width);**

**var pixelHeight = Math.abs(( ybottom - ytop ) / height);**

**pixelSize = Math.min(pixelWidth,pixelHeight);**

**g.scale( width / (xright-xleft), height / (ybottom-ytop) );**

**g.translate( -xleft, -ytop );**

**}**

***//------------------ Animation framework ------------------------------***

**var running = false; *// This is set to true when animation is running***

**function frame() {**

**if (running) {**

***// Draw one frame of the animation, and schedule the next frame.***

**updateFrame();**

**draw();**

**requestAnimationFrame(frame);**

**}**

**}**

**function doAnimationCheckbox() {**

**var shouldRun = document.getElementById("animateCheck").checked;**

**if ( shouldRun != running ) {**

**running = shouldRun;**

**if (running)**

**requestAnimationFrame(frame);**

**}**

**}**

***//----------------------- initialization -------------------------------***

**function init() {**

**canvas = document.getElementById("thecanvas");**

**if (!canvas.getContext) {**

**document.getElementById("message").innerHTML = "ERROR: Canvas not supported";**

**return;**

**}**

**graphics = canvas.getContext("2d");**

**document.getElementById("animateCheck").checked = false;**

**document.getElementById("animateCheck").onchange = doAnimationCheckbox;**

**draw();**

**}**

**</script>**

**</head>**

**<body onload="init()" style="background-color:#EEEEEE">**

**<noscript>**

**<p><b style="color:red">Error: This page requires JavaScript, but it is not available.</b></p>**

**</noscript>**

**<p id="message"><label><input type="checkbox" id="animateCheck"><b>Run the Animation</b></label></p>**

**<div style="float:left; border: 2px solid black">**

**<canvas id="thecanvas" width="800" height="600" style="display:block"></canvas>**

**</div>**

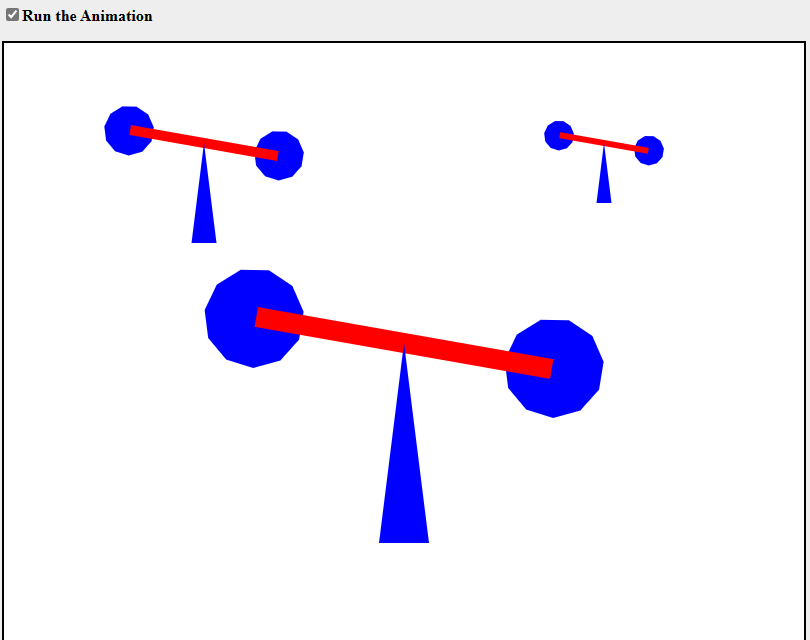
**</body>**

**</html>**

1. **Link do zdalnego repozytorium:**

* https://github.com/Terminalk/GKLab

1. **Wynik działania:** 
   1. **Wynik działania**

****

### 

### 

**Podpunkt b (tworząc graf sceny):**

1. **Wprowadzane dane:**
   1. **Rysowanie wiatraków**

**world = new CompoundObject(); *// Root node for the scene graph.***

***// 1***

**let px = 0;**

**let py = 0;**

**let scale = 1;**

**kolo\_1\_1 = new TransformedObject(filledPolygon);**

**kolo\_1\_1.setColor("blue").setTranslation(1.5 \* scale + px, -0.25 \* scale + py).setScale(scale, scale);**

**world.add(kolo\_1\_1);**

**kolo\_2\_1 = new TransformedObject(filledPolygon);**

**kolo\_2\_1.setColor("blue").setTranslation(-1.5 \* scale + px, 0.25 \* scale + py).setScale(scale, scale);**

**world.add(kolo\_2\_1);**

**linia\_1 = new TransformedObject(filledRect);**

**linia\_1.setScale(3 \* scale, 0.2 \* scale).setColor("red").setRotation(-10).setTranslation(px, py);**

**world.add(linia\_1);**

**podstawa\_1 = new TransformedObject(filledTriangle);**

**podstawa\_1.setScale(0.5 \* scale, 2 \* scale).setColor("blue").setTranslation(0 \* scale + px, -2 \* scale + py);**

**world.add(podstawa\_1);**

***// 2***

**px = -2;**

**py = 2;**

**scale = 0.5;**

**kolo\_1\_2 = new TransformedObject(filledPolygon);**

**kolo\_1\_2.setColor("blue").setTranslation(1.5 \* scale + px, -0.25 \* scale + py).setScale(scale, scale);**

**world.add(kolo\_1\_2);**

**kolo\_2\_2 = new TransformedObject(filledPolygon);**

**kolo\_2\_2.setColor("blue").setTranslation(-1.5 \* scale + px, 0.25 \* scale + py).setScale(scale, scale);**

**world.add(kolo\_2\_2);**

**linia\_2 = new TransformedObject(filledRect);**

**linia\_2.setScale(3 \* scale, 0.2 \* scale).setColor("red").setRotation(-10).setTranslation(px, py);**

**world.add(linia\_2);**

**podstawa\_2 = new TransformedObject(filledTriangle);**

**podstawa\_2.setScale(0.5 \* scale, 2 \* scale).setColor("blue").setTranslation(0 \* scale + px, -2 \* scale + py);**

**world.add(podstawa\_2);**

***// 3***

**px = 2;**

**py = 2;**

**scale = 0.3;**

**kolo\_1\_3 = new TransformedObject(filledPolygon);**

**kolo\_1\_3.setColor("blue").setTranslation(1.5 \* scale + px, -0.25 \* scale + py).setScale(scale, scale);**

**world.add(kolo\_1\_3);**

**kolo\_2\_3 = new TransformedObject(filledPolygon);**

**kolo\_2\_3.setColor("blue").setTranslation(-1.5 \* scale + px, 0.25 \* scale + py).setScale(scale, scale);**

**world.add(kolo\_2\_3);**

**linia\_3 = new TransformedObject(filledRect);**

**linia\_3.setScale(3 \* scale, 0.2 \* scale).setColor("red").setRotation(-10).setTranslation(px, py);**

**world.add(linia\_3);**

**podstawa\_3 = new TransformedObject(filledTriangle);**

**podstawa\_3.setScale(0.5 \* scale, 2 \* scale).setColor("blue").setTranslation(0 \* scale + px, -2 \* scale + py);**

**world.add(podstawa\_3);**

* 1. **Rysowanie jedenastokąta**

**var filledPolygon = new SceneGraphNode(); *// Filled Triangle, width 1, height 1, center of base at (0,0)***

**filledPolygon.doDraw = function (g) {**

**g.beginPath();**

***// g.moveTo(-0.5, 0);***

**let w = 11;**

**let r = 0.5;**

**let kat = 360 / w;**

**for (let i = 1; i <= w; i++) {**

**let rad = ((kat \* i) \* (2 \* Math.PI)) / 360**

**let y = Math.sin(rad) \* r;**

**let x = Math.cos(rad) \* r;**

**g.lineTo(x, y);**

**}**

**g.closePath();**

**g.fill();**

**}**

1. **Wykorzystane komendy:**
   1. **kod źródłowy**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="UTF-8">**

**<script>**

**var canvas; // The canvas that is used as the drawing surface**

**var graphics; // The 2D graphics context for drawing on the canvas.**

**var X\_LEFT = -4; // The xy limits for the coordinate system.**

**var X\_RIGHT = 4;**

**var Y\_BOTTOM = -3;**

**var Y\_TOP = 3;**

**var BACKGROUND = "white"; // The display is filled with this color before the scene is drawn.**

**var pixelSize; // The size of one pixel, in the transformed coordinates.**

**var frameNumber = 0; // Current frame number. goes up by one in each frame.**

**var world; // A SceneGraphNode representing the entire scene.**

**// TODO: Define global variables to represent animated objects in the scene.**

**var rotatingRect; // (DELETE THIS EXAMPLE)**

**/\*\***

**\* Builds the data structure that represents the entire picture.**

**\*/**

**function createWorld() {**

**world = new CompoundObject(); *// Root node for the scene graph.***

***// TODO: Create objects and add them to the scene graph.***

**rotatingRect = new TransformedObject(filledRect); *// (DELETE THIS EXAMPLE)***

**rotatingRect.setScale(2,2).setColor("red");**

**world.add(rotatingRect);**

**}**

**/\*\***

**\* This method is called just before each frame is drawn. It updates the modeling**

**\* transformations of the objects in the scene that are animated.**

**\*/**

**function updateFrame() {**

**frameNumber++;**

***// TODO: Update state in preparation for drawing the next frame.***

**rotatingRect.setRotation(frameNumber\*0.75); *// (DELETE THIS EXAMPLE)***

**}**

**//------------------- A Simple Scene Object-Oriented Scene Graph API ----------------**

**/\*\***

**\* The (abstract) base class for all nodes in the scene graph data structure.**

**\*/**

**function SceneGraphNode() {**

**this.fillColor = null; *// If non-null, the default fillStyle for this node.***

**this.strokeColor = null; *// If non-null, the default strokeStyle for this node.***

**}**

**SceneGraphNode.prototype.doDraw = function(g) {**

***// This method is meant to be abstract and must be OVERRIDDEN in an actual***

***// object. It is not meant to be called; it is called by draw().***

**throw "doDraw not implemented in SceneGraphNode"**

**}**

**SceneGraphNode.prototype.draw = function(g) {**

***// This method should be CALLED to draw the object It should NOT***

***// ordinarily be overridden in subclasses.***

**graphics.save();**

**if (this.fillColor) {**

**g.fillStyle = this.fillColor;**

**}**

**if (this.strokeColor) {**

**g.strokeStyle = this.strokeColor;**

**}**

**this.doDraw(g);**

**graphics.restore();**

**}**

**SceneGraphNode.prototype.setFillColor = function(color) {**

***// Sets fillColor for this node to color.***

***// Color should be a legal CSS color string, or null.***

**this.fillColor = color;**

**return this;**

**}**

**SceneGraphNode.prototype.setStrokeColor = function(color) {**

***// Sets strokeColor for this node to color.***

***// Color should be a legal CSS color string, or null.***

**this.strokeColor = color;**

**return this;**

**}**

**SceneGraphNode.prototype.setColor = function(color) {**

***// Sets both the fillColor and strokeColor to color.***

***// Color should be a legal CSS color string, or null.***

**this.fillColor = color;**

**this.strokeColor = color;**

**return this;**

**}**

**/\*\***

**\* Defines a subclass, CompoundObject, of SceneGraphNode to represent**

**\* an object that is made up of sub-objects. Initially, there are no**

**\* sub-objects. Objects are added with the add() method.**

**\*/**

**function CompoundObject() {**

**SceneGraphNode.call(this); *// do superclass initialization***

**this.subobjects = []; *// the list of sub-objects of this object***

**}**

**CompoundObject.prototype = new SceneGraphNode(); // (makes it a subclass!)**

**CompoundObject.prototype.add = function(node) {**

**this.subobjects.push(node);**

**return this;**

**}**

**CompoundObject.prototype.doDraw = function(g) {**

***// Just call the sub-objects' draw() methods.***

**for (var i = 0; i < this.subobjects.length; i++)**

**this.subobjects[i].draw(g);**

**}**

**/\*\***

**\* Define a subclass, TransformedObject, of SceneGraphNode that**

**\* represents an object along with a modeling transformation to**

**\* be applied to that object. The object must be specified in**

**\* the constructor. The transformation is specified by calling**

**\* the setScale(), setRotate() and setTranslate() methods. Note that**

**\* each of these methods returns a reference to the TransformedObject**

**\* as its return value, to allow for chaining of method calls.**

**\* The modeling transformations are always applied to the object**

**\* in the order scale, then rotate, then translate.**

**\*/**

**function TransformedObject(object) {**

**SceneGraphNode.call(this); *// do superclass initialization***

**this.object = object;**

**this.rotationInDegrees = 0;**

**this.scaleX = 1;**

**this.scaleY = 1;**

**this.translateX = 0;**

**this.translateY = 0;**

**}**

**TransformedObject.prototype = new SceneGraphNode(); // (makes it a subclass!)**

**TransformedObject.prototype.setRotation = function(angle) {**

***// Set the angle of rotation, measured in DEGREES. The rotation***

***// is always about the origin.***

**this.rotationInDegrees = angle;**

**return this;**

**}**

**TransformedObject.prototype.setScale = function(sx, sy) {**

***// Sets scaling factors.***

**this.scaleX = sx;**

**this.scaleY = sy;**

**return this;**

**}**

**TransformedObject.prototype.setTranslation = function(dx,dy) {**

***// Set translation mounts.***

**this.translateX = dx;**

**this.translateY = dy;**

**return this;**

**}**

**TransformedObject.prototype.doDraw = function(g) {**

***// Draws the object, with its modeling transformation.***

**g.save();**

**if (this.translateX != 0 || this.translateY != 0) {**

**g.translate(this.translateX, this.translateY);**

**}**

**if (this.rotationInDegrees != 0) {**

**g.rotate(this.rotationInDegrees/180\*Math.PI);**

**}**

**if (this.scaleX != 1 || this.scaleY != 1) {**

**g.scale(this.scaleX, this.scaleY);**

**}**

**this.object.draw(g);**

**g.restore();**

**}**

**// Create some basic shapes as custom SceneGraphNode objects.**

**var line = new SceneGraphNode(); // Line from (-0.5,0) to (0.5,0)**

**line.doDraw = function(g) {**

**g.beginPath();**

**g.moveTo(-0.5,0);**

**g.lineTo(0.5,0);**

**g.stroke();**

**}**

**var filledRect = new SceneGraphNode(); // Filled square, size = 1, center = (0,0)**

**filledRect.doDraw = function(g) {**

**g.fillRect(-0.5,-0.5,1,1);**

**}**

**var rect = new SceneGraphNode(); // Stroked square, size = 1, center = (0,0)**

**rect.doDraw = function(g) {**

**g.strokeRect(-0.5,-0.5,1,1);**

**}**

**var filledCircle = new SceneGraphNode(); // Filled circle, diameter = 1, center = (0,0)**

**filledCircle.doDraw = function(g) {**

**g.beginPath();**

**g.arc(0,0,0.5,0,2\*Math.PI);**

**g.fill();**

**}**

**var circle = new SceneGraphNode();// Stroked circle, diameter = 1, center = (0,0)**

**circle.doDraw = function(g) {**

**g.beginPath();**

**g.arc(0,0,0.5,0,2\*Math.PI);**

**g.stroke();**

**}**

**var filledTriangle = new SceneGraphNode(); // Filled Triangle, width 1, height 1, center of base at (0,0)**

**filledTriangle.doDraw = function(g) {**

**g.beginPath();**

**g.moveTo(-0.5,0);**

**g.lineTo(0.5,0);**

**g.lineTo(0,1);**

**g.closePath();**

**g.fill();**

**}**

**// ------------------------------- graphics support functions --------------------------**

**/\*\***

**\* Draw one frame of the animation. Probably doesn't need to be changed,**

**\* except maybe to change the setting of preserveAspect in applyLimits().**

**\*/**

**function draw() {**

**graphics.save(); *// to make sure changes don't carry over from one call to the next***

**graphics.fillStyle = BACKGROUND; *// background color***

**graphics.fillRect(0,0,canvas.width,canvas.height);**

**graphics.fillStyle = "black";**

**applyLimits(graphics,X\_LEFT,X\_RIGHT,Y\_TOP,Y\_BOTTOM,false);**

**graphics.lineWidth = pixelSize; *// Use 1 pixel as the default line width***

**world.draw(graphics);**

**graphics.restore();**

**}**

**/\*\***

**\* Applies a coordinate transformation to the graphics context, to map**

**\* xleft,xright,ytop,ybottom to the edges of the canvas. This is called**

**\* by draw(). This does not need to be changed.**

**\*/**

**function applyLimits(g, xleft, xright, ytop, ybottom, preserveAspect) {**

**var width = canvas.width; *// The width of this drawing area, in pixels.***

**var height = canvas.height; *// The height of this drawing area, in pixels.***

**if (preserveAspect) {**

***// Adjust the limits to match the aspect ratio of the drawing area.***

**var displayAspect = Math.abs(height / width);**

**var requestedAspect = Math.abs(( ybottom-ytop ) / ( xright-xleft ));**

**var excess;**

**if (displayAspect > requestedAspect) {**

**excess = (ybottom-ytop) \* (displayAspect/requestedAspect - 1);**

**ybottom += excess/2;**

**ytop -= excess/2;**

**}**

**else if (displayAspect < requestedAspect) {**

**excess = (xright-xleft) \* (requestedAspect/displayAspect - 1);**

**xright += excess/2;**

**xleft -= excess/2;**

**}**

**}**

**var pixelWidth = Math.abs(( xright - xleft ) / width);**

**var pixelHeight = Math.abs(( ybottom - ytop ) / height);**

**pixelSize = Math.min(pixelWidth,pixelHeight);**

**g.scale( width / (xright-xleft), height / (ybottom-ytop) );**

**g.translate( -xleft, -ytop );**

**}**

**//------------------ Animation framework ------------------------------**

**var running = false; // This is set to true when animation is running**

**function frame() {**

**if (running) {**

***// Draw one frame of the animation, and schedule the next frame.***

**updateFrame();**

**draw();**

**requestAnimationFrame(frame);**

**}**

**}**

**function doAnimationCheckbox() {**

**var shouldRun = document.getElementById("animateCheck").checked;**

**if ( shouldRun != running ) {**

**running = shouldRun;**

**if (running)**

**requestAnimationFrame(frame);**

**}**

**}**

**//----------------------- initialization -------------------------------**

**function init() {**

**canvas = document.getElementById("thecanvas");**

**if (!canvas.getContext) {**

**document.getElementById("message").innerHTML = "ERROR: Canvas not supported";**

**return;**

**}**

**graphics = canvas.getContext("2d");**

**document.getElementById("animateCheck").checked = false;**

**document.getElementById("animateCheck").onchange = doAnimationCheckbox;**

**createWorld();**

**draw();**

**}**

**</script>**

**</head>**

**<body onload="init()" style="background-color:#EEEEEE">**

**<noscript>**

**<p><b style="color:red">Error: This page requires JavaScript, but it is not available.</b></p>**

**</noscript>**

**<p id="message"><label><input type="checkbox" id="animateCheck"><b>Run the Animation</b></label></p>**

**<div style="float:left; border: 2px solid black">**

**<canvas id="thecanvas" width="800" height="600" style="display:block"></canvas>**

**</div>**

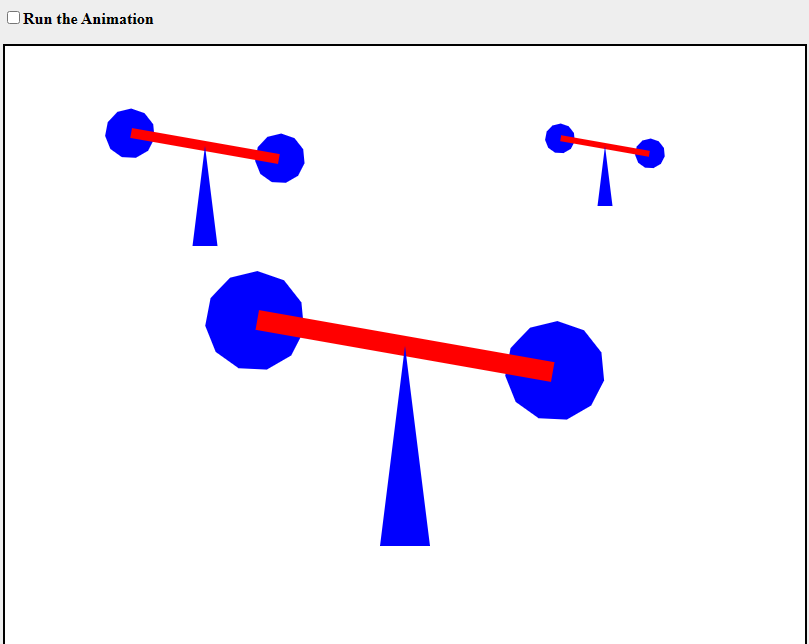
**</body>**

**</html>**

1. **Link do zdalnego repozytorium:**

* https://github.com/Terminalk/GKLab

1. **Wynik działania:**
   1. **Wynik działania programu:**

****

1. **Wnioski:**

**Tworzenie obiektów 2d jest łatwiejsze za pomocą metody tworzenia grafu sceny, w języku JavaScript oraz obiektu Canvas jesteśmy w łatwy sposób tworzyć proste grafiki 2d.**