3D Transformation

Rotation: In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling: three coordinates are used. Let us assume that the original coordinates are (X, Y, Z), scaling factors are (S_x, S_y, S_z) respectively, and the produced coordinates are (X', Y', Z').

This can be mathematically represented as shown below:

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad P' = P \cdot S \quad [X' Y' Z' 1] = [X Y Z 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [X S_x, Y S_y, Z S_z, 1]$$

What is a Shader?: It's simply a program that runs in the graphics pipeline and tells the computer **how to render each pixel**.
They are called shaders because they're often **used to control lighting and shading effects**.

A shader's sole purpose is to **return four numbers: r, g, b** and a **alpha**, defines the transparency

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{ fragColor = vec4(1.0, 0.0, 0.0, 1.0); }
```

This function runs for **every single pixel** on screen. It returns those four color values, and that becomes the color of the pixel. This is what's called a pixel shader (sometimes referred to as a fragment shader).

Shader Inputs fragCoord tells the pixel's x and y (and z, if you're working in 3D)

Note: for any vec4, you can access its component via obj.x, obj.y, obj.z and obj.w or via obj.r, obj.g, obj.b, obj.a.

Screen size is not a built in variable like pixel location was. But we can still send this information from the **CPU** (the main program) to the **GPU** (my shader)

Vertex Shader: handles the **processing of individual vertices**.
They are fed vertex attribute data, as specified from a vertex array object by a drawing command.

A vertex shader receives a single vertex from the vertex stream and generates a single vertex to the output vertex stream. There must be a mapping (1 1) from input vertices to output vertices.

Vertex shaders typically perform transformations to post-projection space. They can also be used to do per-vertex lighting, or to perform setup work for later shader stages.

Vertex Post-Processing: a stage in the rendering pipeline where the vertex outputs of the vertex processing undergo a variety of operations. Many of these are setup for Primitive Assm. and Rasterization stages.

operations include: for example:
- Transformation Feedback (a way of recording values output from the vertex processing stage into Buffer Objects.
- Clipping (to the view volume)
- Perspective Divide
- Viewport transform

Primitive Assembly: a stage where Primitives are divided into sequence of individual base primitives. After minor processing, they are passed along to the rasterizer to be rendered.

The purpose of the primitive assembly step is to convert a vertex stream into a sequence of base primitives.
For example, a primitive which is a line list of 12 vertices needs to generate 11 line base primitives.

Example Vertex Shader:

```
PosTexLi SimpleVS(PosNorTexInput){
PosTexLi output = (PosTexLi)0;        // initialization
output.Pos = mul(Input.pos, g_worldViewProjection); // position from object space to
output.Tex = Input.Tex;                              homogenious clip space
output.normal = normalize(mul(Input.Nor, g_world)).xyz);
output.Li = saturate(dot(output.normal, g_LightDir.xyz));

return output; }
```

Outputs: are passed to the next section of the pipeline. They are in this order.
1. Tessellation
2. Geometry Shader
3. Vertex Post-Processing

Pixel Shader (also known as fragment shader):

compute color and other attributes of each "fragment" - a technical term usually meaning a single pixel. They usually output one screen pixel as a color value.

Pixel shaders range from always outputing the same color, to applying a lighting value, to doing bump mapping etc.

main todo:
- (simple) Texturing
- (simple) Normal Mapping
- (simple) lighting

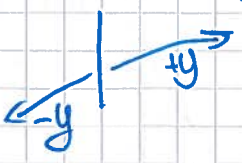
Example of a Pixel Shader:

```
float4 simplePS(PosTex Input): SV_Target0{
  float4 matDiffuse = g_Diffuse.Sample(samLinearClamp, Input.Tex);
  return float4(matDiffuse.rgb * Input.Li, 1); }
```

Terrain Vertex Shader

```
PosTex TerrainVS(uint VertexID : SV_VertexID){
PosTex output = (PosTex)0;                    // initialization
int x = (float) VertexID % (float) g_TerrainRes;
int z = VertexID / g_TerrainRes;

output.Pos.x = ((float)x / g_TerrainRes - 0.5f);
output.Pos.y = g_HeightMap[VertexID];
output.Pos.z = ((float)z / g_TerrainRes - 0.5f);
output.Pos.w = 1.0f;

output.Tex.x = ((float)x / g_TerrainRes);
output.Tex.y = ((float)z / g_TerrainRes);

output.Pos = mul(output.Pos, g_WorldViewProjection);
return output;
}
```

along +y-axis.
↳ means of look into the direction +y



Pixel Shader:
- Vertex shaders work on vertices
- pixel shaders work on pixels.

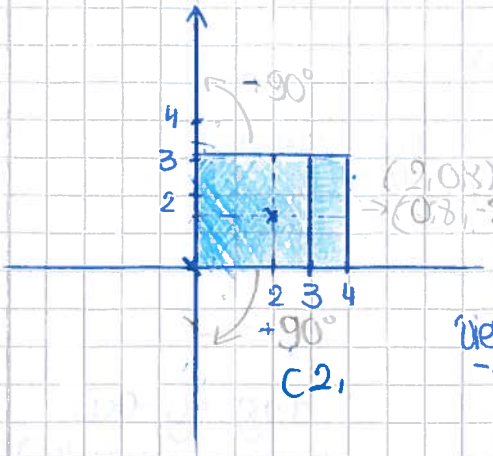
Before a pixel is written to the framebuffer, it is first passed through the pixel shader.

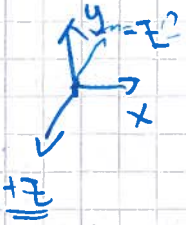
needs to output a single color. The value of this color may be computed in several ways, taking into account the texture, ambient light ... .

The inputs used for this by the shader are attributes coming from the vertex texture, shader parameters coming from the application, and texture data coming from the texture samplers.

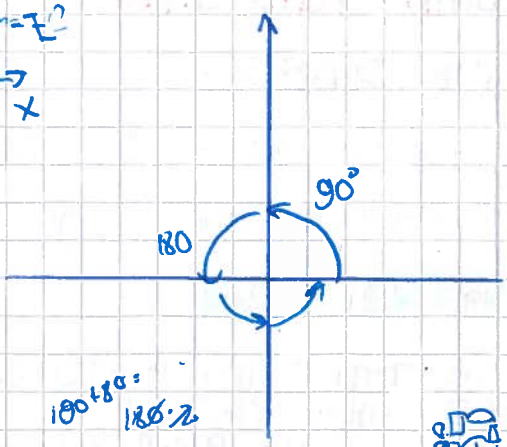
Transformations

```
XMMATRIX matTrans1 = XMMATRIXTRANSLATION(3,0,0);
XMMATRIX matScale1 = XMMatrixScaling(1,1,2);
"        matScale2:  "  Scaling(1, 2.0f/3.0f, 1);

matRotY1 = Rotation -90°
matRotY2 = Rotation +90°

matFinal = matRotY1 * matTrans1 * matRotY2 * matScale1 * matScale2;
```



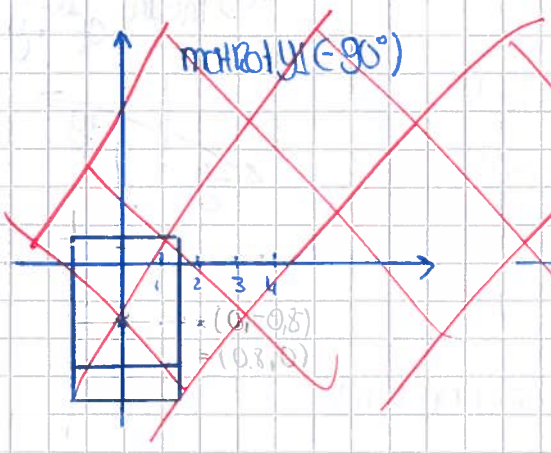


view along -z

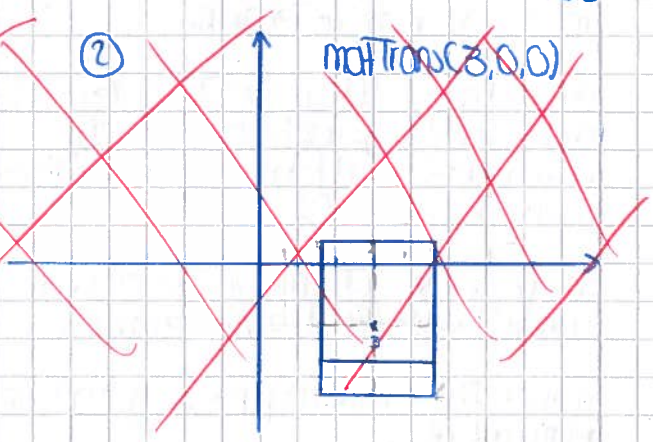


180+180=
180:2

① matRotY1(-90°)



② matTrans(3,0,0)



(5,7)
↳ 90°
(7,-5)

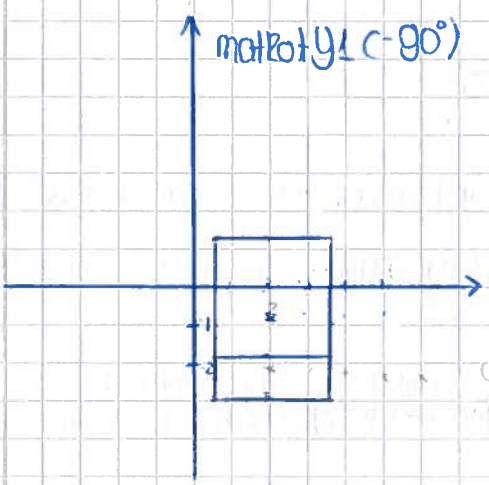
(+2,5)
↳ 90°)
(-5,-2)

Anticlockwise:

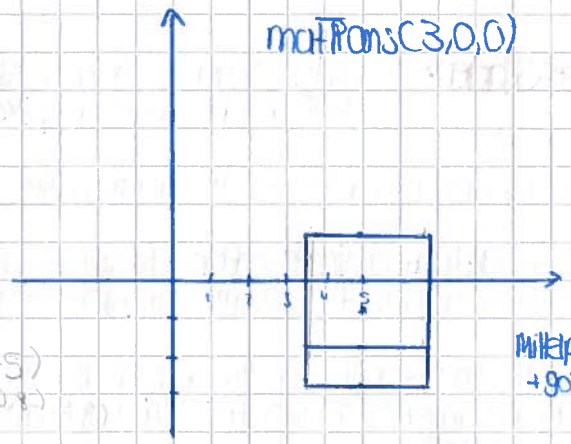
(2,3)
(-3,2)

(-5,-7)
(7,-5)

matRotY1(-90°)

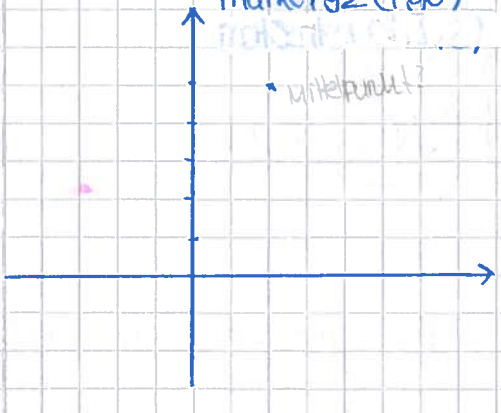


matTrans(3,0,0)

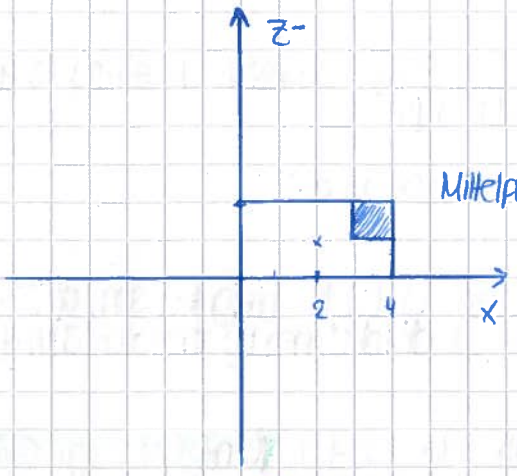


Mittelpunkt (5;0,9)
+90° (+0,9;5))

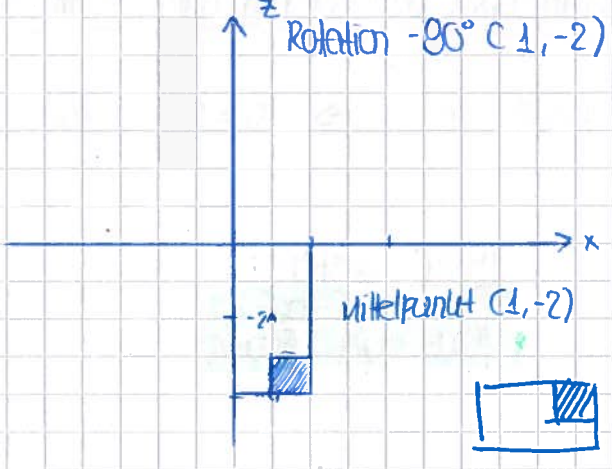
matRotY2(+90°)



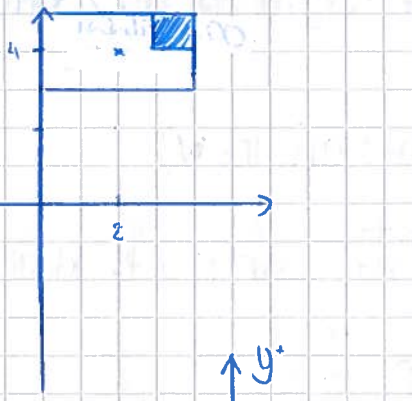
Mittelpunkt?

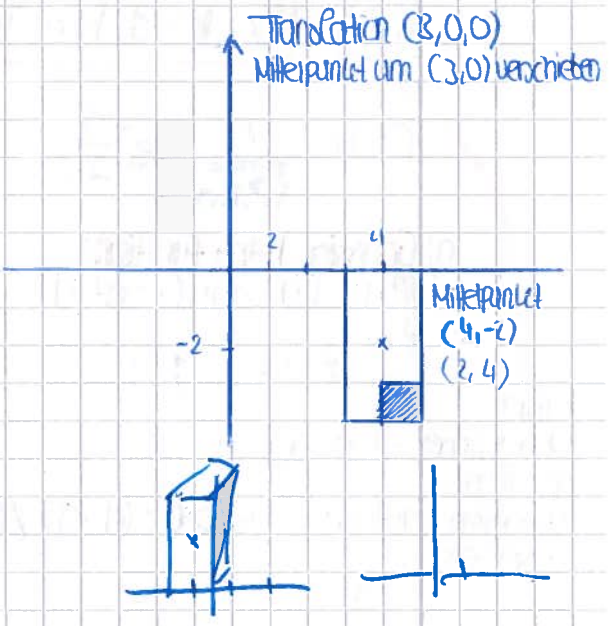


Mittelpunkt (2,1) → (-1,2) → (3,0)
-90° (2,2) → 90°
↳ x(-2,2)

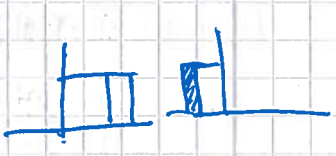


Rotation -90° (1,-2)
Mittelpunkt (1,-2)

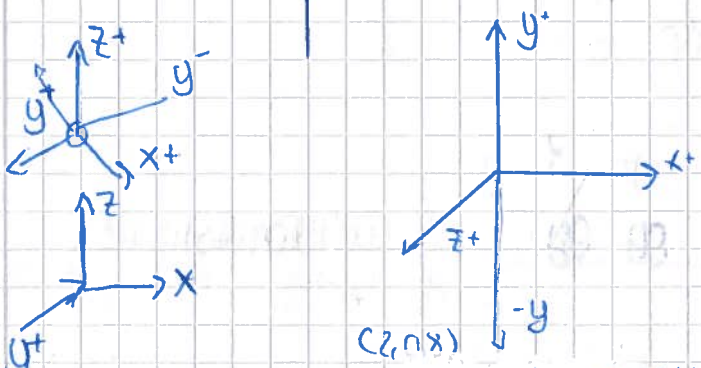


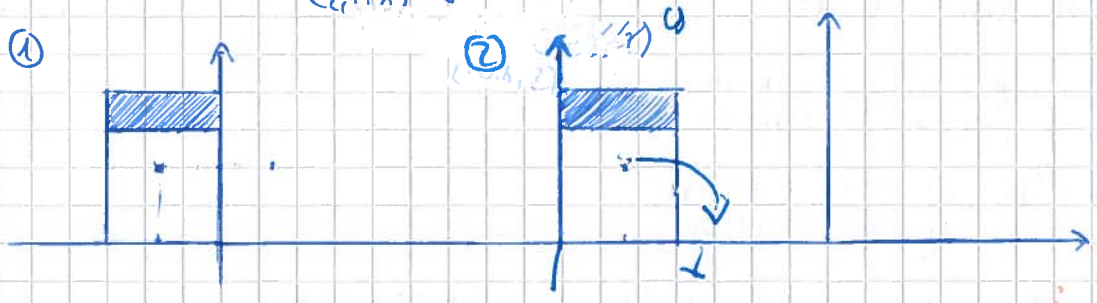


Translation (3,0,0)
Mittelpunkt um (3,0) verschieben
Mittelpunkt (4,-2) (2,4)



View along +y:





3) Array Filtering 2D

To-Do: implement a method, that realizes a 5×5 Gaussian smoothing operation on the 2D-texture stored in the array in.

The smoothed texture should be stored in the given array out.

input texture is of size w×w. (size of in) (sum of the 25 weighted values!)
Smoothing operation: filter replaces each value v_i by the weighted sum of 25 values v_j (i.e. from itself and the 24 surrounding pixels), divided then by the total sum of the weights.

Each weight w_j is computed by calling the helper function **float Gaussian(float d)** with the Euclidian distance between the pixel centers corresponding to the values v_i and v_j.

$$G(d) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{d^2}{2\sigma^2}}$$ → als Potenz schreiben → $\sigma = 2$, distance d

a) **Gaussian helper function:**

```
float Gaussian(float d)
{
    float pi = ...;
    float sigma = 2.0f;
    return exp(-(d*d)/(2.0f*sigma*sigma)) / sqrt(2.0f*pi*sigma*sigma);
}
```

Zahlen lieber immer als float-Variablen schreiben!

Useful functions:
float exp(float x)
float sqrt(float x)

b)
```
void filterArray(float* in, float* out, int w)
{
    for (int ix=0; ix<w; ix++)
    for (int iy=0; iy<w; iy++)
    {
        float colorSum = 0.0f;
        float weightSum = 0.0f;
        for (int gx=-r; gx<=r; gx++)
        for (int gy=-r; gy<=r; gy++) {
            float dist = sqrt(gx*gx + gy*gy);   // Euclidian distance
            float weight = Gaussian(dist);
            weightSum += weight;
            if ((gx+ix)<0) continue;
            if ((gy+iy)<0) continue;
            if ((gx+ix)>=w) continue;
            if ((gy+iy)>=w) continue;
            int idx = (ix+gx)+(iy+gy)*w;
            colorSum += in[idx] * weight;
        }
        out[ix+iy*w] = colorSum / weightSum;
    }
}
```

because r can be negative

cbuffer CBufferPerObject
{ float4x4 WorldViewProjection : WORLDVIEWPROJECTION; }

Constant Buffers:

Purpose: organize one or more shader constants.

Shader constant: input which the CPU sends to a shader, which remains constant for all the primitives processed by a single draw call.

WorldViewProjection → transforms vertices from object space → world space
view space → homogeneous CP
in a single transformation.

WORLDVIEWPROJECTION: known as semantic and is a hint to the CPU-side application about the intended use of the variable

→ use is not optional, must be associated with a shader constant for the effect to receive updates to the concatenated WVP matrix each frame.

Render State — Input Assembler stage is the entry point of the graphics pipeline where we supply vertex and index data for the objects we want to render.

We can customize the nonprogrammable stages of the Direct3D pipeline
for example: the rasterizer stage is customized through a RasterizerState Object.

RasterizerState DisableCulling { CullMode = NONE; };

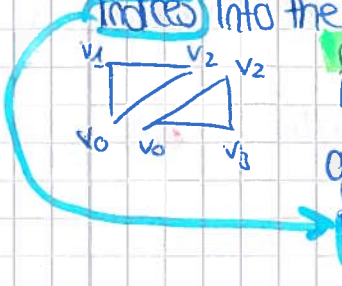
Vertex Buffers: a vertex is at least a position in three-dimensional space
↳ might also contain a normal (useful for light calculations), color, texture coordinates and much more

all this data is available for the Input Assembler Stage through the vertex buffer

Index Buffer: (optional) second type of input into the IA stage.
identify specific vertices in the vertex buffer and are employed to reduce duplication of used vertices.

Example: Rendering a quadrilateral → two triangles
we have now six total vertices, with two of the vertices duplicated.

With an index buffer, we can instead specify just the four unique vertices and six indices into the vertex buffer.



duplicate vertices: v_0, v_2
instead of writing: $v_0\ v_0\ v_1\ v_2\ v_2\ v_3$ we'll have = $v_0\ v_1\ v_2\ v_3$ in our vertex buffer

and Index Buffer will have the connections of each vertices to a triangle

Primitive Types

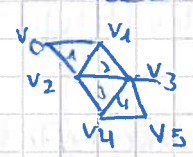
We must define, how the input-assembler stage should interpret the vertices.

line list: connects pairs of points into line segment (not connected with other line segments)

line strip: connected
triangle list: each set of three vertices is interpreted as an isolated triangle
shared vertices are repeated

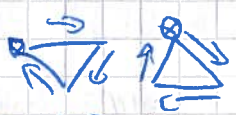
triangle strip: connected triangles, shared vertices are not repeated.

↳ the numbering of the vertices does not make clear how groups of three vertices are rendered.



Direct3D draws the triangles in the following orders

① $V_0 V_1 V_2$ ② $V_1 V_3 V_2$ ③ $V_2 V_3 V_4$ ④ $V_3 V_5 V_4$



starting from the top

Direct3D winds triangles in clockwise order to aid in backface culling.

Primitives with Adjacency: we specify vertices (adjacent vertices) "surrounding" the base primitive. (adjacency data)

Tessellation Stages: adds detail to objects directly on the GPU. (adjust level of detail)

Hardware tessellation enables us to subdivide an object dynamically and without the cost of additional geometry passed to the input-assembler stage.

HS and DS are programmable, the tessellator stage is not.

Geometry Shader Stage (GS):
- operate on complete primitives.
- have the ability to completely remove or add geometry from the pipeline

↳ we can create a particle effect system → In the geometry shader, we can create quads around those central points. (point sprites)

Rasterizer Stage: converts primitive into a raster image (also known as bitmap)

↳ raster image: two dimensional array of pixels (colors)

↳ determines what pixels should be rendered and passes those pixels to the pixel shader stage

④ also passes per-vertex values interpolated across each primitive.

Output-Merger Stage (OM)

produces the final rendered pixel (not programmable)
determines which pixels are visible in the final render through depth testing and stencil testing.

depth testing: makes use of the distance between the object and the camera for each corresponding pixel written to the render target.

If the pixel already in the render target is closer to the camera than the pixel being considered, the new pixel is discarded.

Backface culling discards primitives which are not facing the camera

stencil = StateRange

SV_Target semantic: indicates that the output will be stored in the render target bound to the om-stage.

render target is a texture mapped to the screen and is known as the back buffer.

4) Shading and Lighting

To-Do: shader, that evaluates the Phong lighting model for a given point light source at the incoming fragments of type shaderValues and writes the resulting color to the bound backbuffer.

reflection vector is computed with the following equation:

$R = 2 \cdot (N \circ L) \cdot N - L$ $\quad N$: surface normal, L: light vector

should've given you the hint of a Pixel Shader!

```
VS_OUTPUT VertexShader(VS_INPUT IN) // reflection modeling!
{
    VS_OUTPUT OUT = (VS_OUTPUT)0;
    OUTPUT.Position = mul(IN.ObjectPosition, WorldViewPosition);   // from object space to world space
    // computing texture coordinates
    OUTPUT.Normal = normalize(mul(float4(IN.Normal, 0), World).xyz);   // converting normal vector to world space (has only xyz comp.)
        ↳ new vector which was the result of the multiplication between IN.Normal and World has to be normalized again, so that it will be a normal again
    OUT.LightDirection = normalize(LightDirection); // g_LightDir xyz has already been given to us
    float3 worldPosition = mul(IN.ObjectPosition, World).xyz;
    OUT.ViewDirection = normalize(CameraPosition - worldPosition);   // never forget to transform!
        ViewDirection is calculated by subtraction
    return OUT; }
```

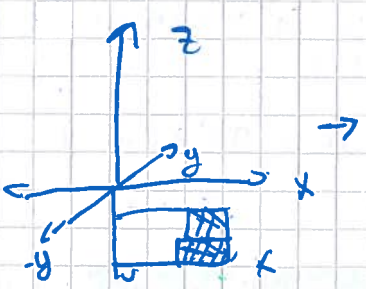
our version is completely different, because we had different structs

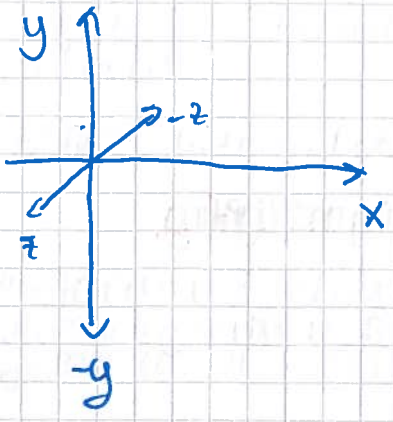
actual solution: $k_d \cdot (\vec{l} \cdot \vec{n})\, I_i(x,we) + k_s \cdot (\vec{r} \cdot \vec{v})^n \cdot I_i(x,we) + k_a \cdot I_a$

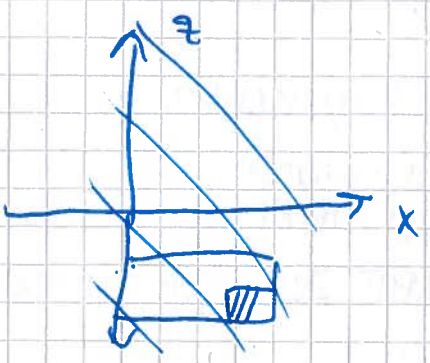
```
float4 PSCin ShaderValues frag) : SV_Target0 {
    float3 n = normalize(frag.normal);
    float3 p = frag.worldPos;
    float3 l = p - (g_light);   // point position of a point light in world space (l;p) => were creating a light vector here!
① float dist = length(l);
① float att = 1.0 / (dist * dist * g_att);   // for calculating the attenuation factor of the light in dependence of the distance
    l = normalize(l);
    float3 v = p - g_cam;
    float d = g_diffuse * dot(n, l);
    float3 r = reflect(l, n);
    float s = g_specular * pow(dot(v, r), g_specularPow);
    float3col = frag.color * (g_ambient + d + s) * att;
```

Note: $A = (a_1 | a_2)$, $B = (b_1 | b_2)$, $\overrightarrow{AB} = \begin{pmatrix} b_1 - a_1 \\ b_2 - a_2 \end{pmatrix}$

① ② $f_{att}(d) = \frac{1}{g\_att \cdot d^2}$

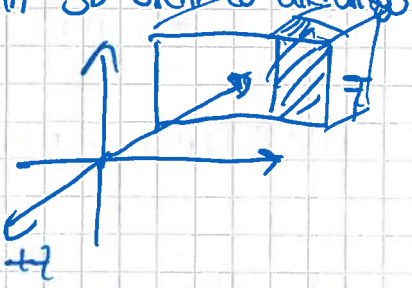


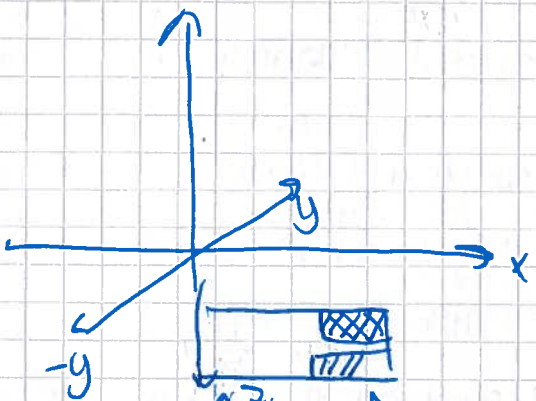


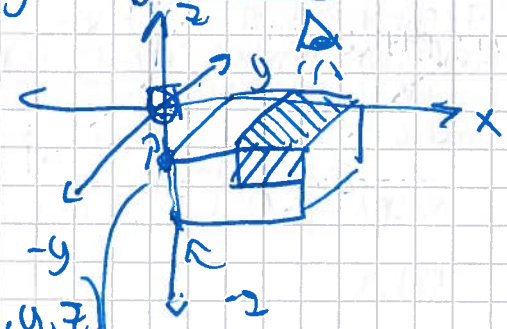


① wir sehen in der 2Dimensionalen Perspektive:

in 3D sieht es allerdings so aus:

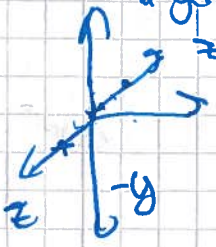




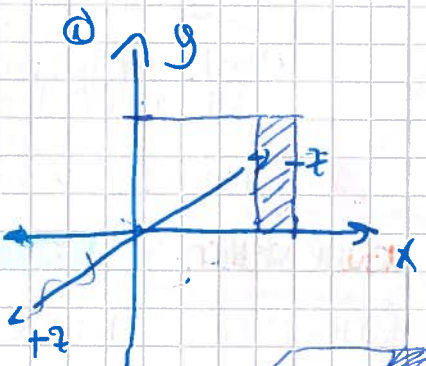


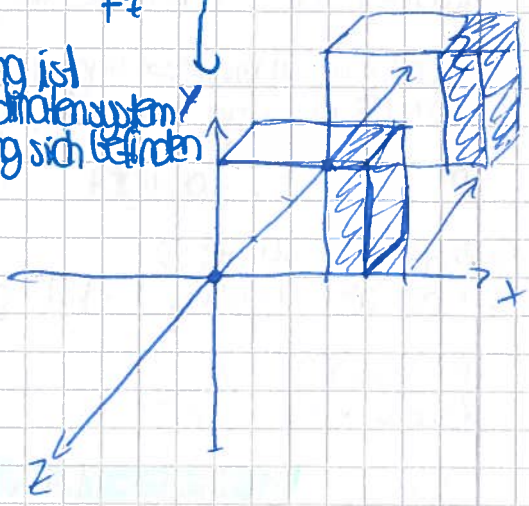
(x, y, z)

alles was hinter dem Ursprung ist wird auf den rechten Koordinatensystem a genau auf dem Ursprung sich befinden → siehe ①



①





Sprite Rendering

Vertex is send through graphics pipeline with PRIMITIVE-POINTLIST which is then converted into a 2D sprite parallel to the view plane using a geometry shader.

Culling of counter-clockwise triangles is enabled.

In the fragment stage (pixel shader), each sprite should be rendered as a tinted black circle; fragments outside that circles should be set to full transparency.

↳ Point Sprite Shader: creates us to render a textured quad (or other shape) while specifying just a single vertex (a point)

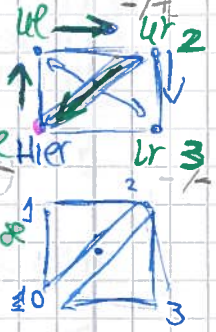
Point represents the center of quad, and the geometry shader constructs the four surrounding vertices to form the quad.

billboarding the quad positions the vertices so that quad always faces the camera (spherical billboarding) & cylindrical billboarding (constraints the rotation around a single axis).

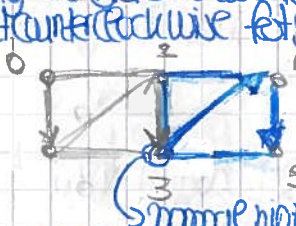
A spherical billboarding Point Sprite Shader

Recurces

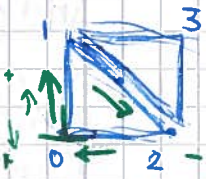
```
static const float2 QuadUVs[4] =
{ float2(0.0f, 1.0f),   // v0, lower-left
  float2(0.0f, 0.0f),   // v1, upper-left
  float2(1.0f, 0.0f),   // v2, upper right
  float2(1.0f, 1.0f)    // v3, lower-right
};
```



Die Ausrichtung der Dreiecke (wo ist vorne?) wird durch die Reihenfolge und das Flag frontCounterClockwise festgelegt.



counterclockwise
damit keine Pixels gehen werden
ungerichtete Strecke hinzugefügt (3,5)
normale hinzugefügt (1,3) und im zweiten Dreieck (2,3) etc.




```
[maxvertexcount(6)]   -> outputs six vertices
void geometry-Shader (point VS-OUTPUT IN[1], inout TriangleStream<GS-OUTPUT> triStream)
{   GS-OUTPUT out = (GS-OUTPUT) 0;
    : adding the vertices into our triStream.
```

-> takes in a single point, constructs four and outputs six total vertices (two triangles in a list, duplicating two of the constructed points)

by default, triangles defined with counter-clockwise vertices are processed as front facing triangles

s. assignment: counter clockwise

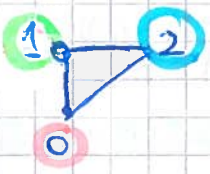
der dritte Punkt wird einfach hinzugefügt. Richtung vom ersten Dreieck ist wichtig!

(*) see the second side

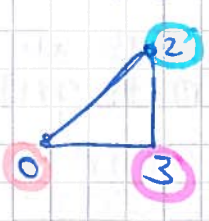
```
float2 halfSize = IN[0].Size / 2.0f;
float3 direction = CameraPosition - IN[0].Position.xyz; //get the vector
float3 right = cross(normalize(direction), CameraUp);

float3 offsetX = halfSize.x * right;
float3 offsetY = halfSize.y * CameraUp;

float4 vertices[4];   // some kind of an array which saves all the vertices
vertices[0] = float4(IN[0].Position.xyz + offsetX - offsetY, 1.0f);
vertices[1] = float4(IN[0].Position.xyz + offsetX + offsetY, 1.0f);
vertices[2] = float4(IN[0].Position.xyz - offsetX + offsetY, 1.0f);
vertices[3] = float4(IN[0].Position.xyz - offsetX - offsetY, 1.0f);
```

creating the 4 vertices from the input point

```
out.Position = mul(vertices[0], viewProjection);
out.TextureCoordinate = QuadUVs[0];
triStream.Append(out);

out.Position = mul(vertices[1], viewProjection);
out.TextureCoordinate = QuadUVs[1];
triStream.Append(out);

out.Position = mul(vertices[2], viewProjection);
out.TextureCoordinate = QuadUVs[2];
triStream.Append(out);

out.Position = mul(vertices[0], viewProjection);
out.TextureCoordinate = QuadUVs[0];
triStream.Append(out);

out.Position = mul(vertices[2], viewProjection);
out.TextureCoordinate = QuadUVs[2];
triStream.Append(out);

out.Position = mul(vertices[3], viewProjection);
out.TextureCoordinate = QuadUVs[3];
triStream.Append(out);
```



→ Speicherung der Vertex-Eigenschaften
→ Transformation (muss immer in Richtung der Kamera stehen, deswegen × viewProjection)
→ TextureCoordinate speichern



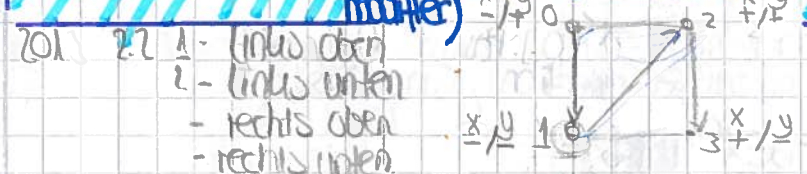
} end of Geometry Shader

Geometry Shader using TriangleStrips:

```
[maxvertexcount(4)]
void geometry_shader(point VS_OUTPUT IN[1], inout TriangleStream<GS_OUTPUT> triStream)
{
    GS_OUTPUT out = (GS_OUTPUT)0;
float2 halfSize = IN[0].Size / 2.0f;
float3 direction = cameraPosition - IN[0].Position.xyz;
float3 right = cross(normalize(direction), cameraUp);

create offset values
create new vertices
```

Shaders parameters: ① defines the type of primitive (point), structure of the input data (VS_OUTPUT), how many vertices will be processed on each

② StreamOutputObject<T> type (is always prefixed with the **inout** modifier)

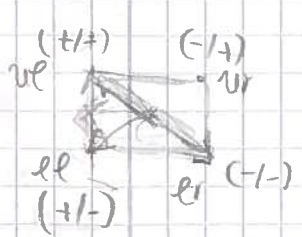



```
1 vertices[0] = float4(IN[0].Position.xyz - offsetx + offsety, 1.0f);
2 vertices[1] = float4(IN[0].Position.xyz - offsetx - offsety, 1.0f);
3 vertices[2] = float4(IN[0].Position.xyz
```

→ bei counterClockwise ist die positive und negative Ausrichtung komplett anders!

Lösung für 2013 2.2

```
void Add(float3 pos, inout TriangleStream<SpriteFragment> s){
    SpriteFragment f;   // create helper variable
    f.pos = mul(float4(pos, 1.0f), viewProjection);
    s.Append(f); }
```



culling for counter-clockwise triangles is enabled →

```
[maxVertexCount(4)]
void SpriteGS(point SpriteVertex v[1], inout TriangleStream<SpriteFragment> s)
{
    float3 p = v[0].pos;
    float3 r = v[0].rad;
    // Bottom left
    Add(p - g_Up*r - g_Right*r, s);
    // Top left
    Add(p + g_Up*r - g_Right*r, s);
    // Bottom right
    Add(p - g_Up*r + g_Right*r, s);
    // Top right
    Add(p + g_Up*r + g_Right*r, s);
```

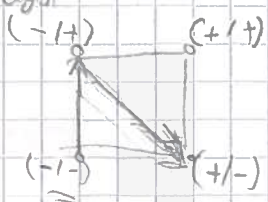
add triangles with vertices which are clockwise

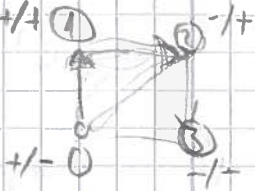
position of the point (equal to IN[0].position)

x-axis

y

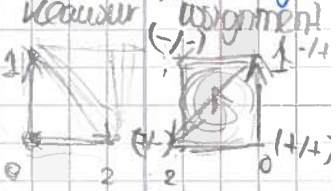
x

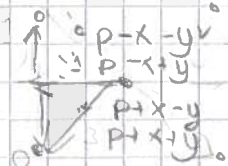


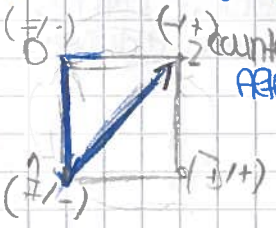


clockwise = normal C-System

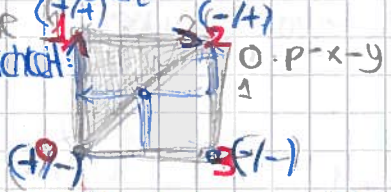
counterclockwise = swapped signs



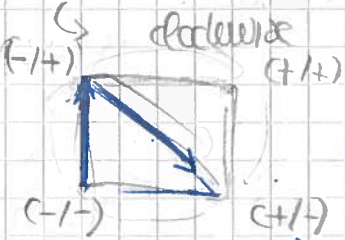

p-x-y
p-x+y
p+x-y
p+x+y

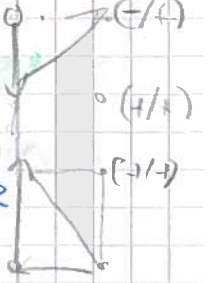


Buch:



- im Uhrzeigersinn gezeichnet
- Dreiecke sind allerdings counterclockwise gerendert
- benutzt daher auch das Koordinatensystem von clockwise Quad, da Richter zu orientieren (macht alles gespiegelt?!)

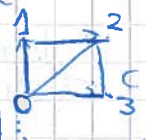




Übungsaufgabe (2013 | 2.2)

In der Angabe steht: Culling of counter-clockwise triangles is enabled.

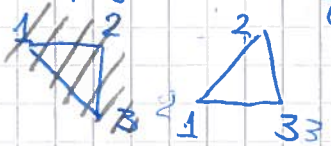
→ Dreiecke werden im Uhrzeigersinn gerendert.

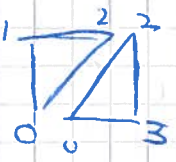


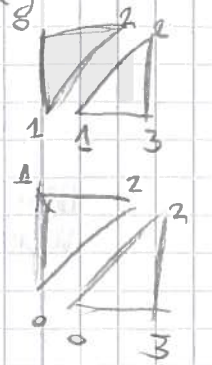
Bei Triangle-Strips gibt es eine Besonderheit:
Culling nach dem ersten Dreieck wird für jedes Dreieck jeweils vertauscht

culling for clockwise

1. Dreieck:







1. Dreieck

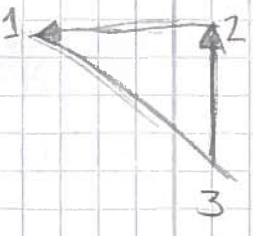
culling of counter-clockwise triangles → draw clockwise

Uebung:

Bottom left	(-/-)
Top Left	(-/+)
Bottom right	(+/-)
Top right	(+/+)

2. Dreieck:

culling of clockwise triangles → draw counterclockwise



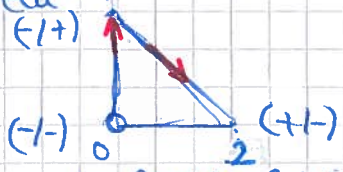
Assignment

Top Left	(-/-)
Bottom Left	
Top right	→ kommentare waren falsch...
Bottom right	

↳ Bottom Left
Top Left
Bottom right
Top right

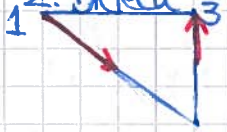
richtige Version

1 Dreieck: 1



culling of counter-clockwise triangles → draw clockwise

2. Dreieck:



culling of clockwise triangles → draw counter clockwise

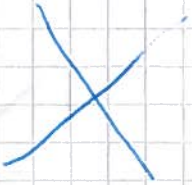
Slide 09. page 08

1. Dreieck.

clockwise

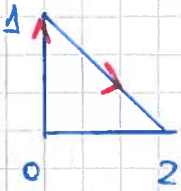
counter clockwise

2 Dreieck:

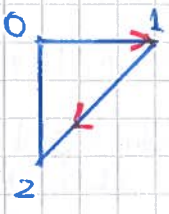


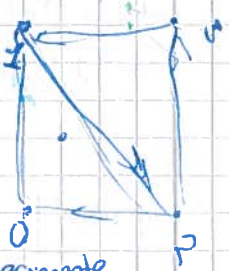
→ drawing triangles clockwise · 2 possibilities:

Uebungs-version.



Assignment version (Slide 09 page 08)

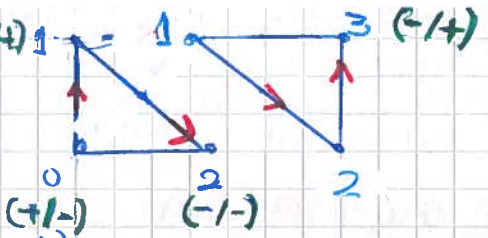




Höhensegmente

Buch:

Bottom Left	0	(+/+)
Top Left	1	(+/+)
Bottom right	2	(-/-)
Top right	3	(-/+)



immer auf die Beschriftung achten

-1,1 1,1

-1,-1 1,-1

(version of the book is somehow swapped)

Hausaufgabe:

```
void Add(float3 pos, float2 tex, inout TriangleStream<QuadVertex> s)
{
    QuadVertex v = mul(float4(pos, 1.0), g_viewProjection);
    v.pos = float4(mul(pos, g_viewProjection), 1.0); // position of QuadVertex is a float4!
    v.tex = tex;

    s.Append(v); }

void SpriteGS(point SpritePoint vertex[1], inout TriangleStream<QuadVertex> s)
{
    // create helper variables for radius and position
    // -> remember assignment 9: radius.x and radius.y
    float3 pos = vertex[0].pos;
    float radius = vertex[0].radius;

    x = g_right,  y = g_up
    (0,1) (1,1)
    (0,0) (1,0)

    // Bottom left
    float2 texture1 = (0,0);
    Add(pos - radius*g_right - radius*g_up, texture1, s);
    // Top left
    float2 texture2 = (0,1);
    Add(pos - radius*g_right + radius*g_up, texture2, s);

    // Bottom right
    float2 texture3 = (1,0);
    Add(pos + radius*g_right - radius*g_up, texture3, s);

    // Top right
    float2 texture4 = (1,1);
    Add(pos + radius*g_right + radius*g_up, texture4, s);
}
```

actual geometry shader

Toon Shading: non-photorealistic rendering technique, which uses discrete levels of intensity instead a continuous intensity distribution.

Objects silhouette are rendered to achieve a comic-style look

durch endliche Intervalle oder Abstände voneinander getrennt

a) Discretize the diffuse light intensity using 5 levels and multiply it with the input color.

```
int diffuse_intensity_int = diffuse_intensity * 5.0f;
float diffuse_intensity_discretized = (float) intensity_int / 5.0f;
```

b) Render every fragment in a specular highlight (spec > 0.5) in white color. Don't affect the alpha channel of the input color.

c) Render the black silhouettes. Defined as the set of points at which the surface normal ($\vec{n}$) and the view direction ($\vec{v}$) are nearly perpendicular to each other ($\vec{v} \cdot \vec{n} \approx 0$).

To render equal thickness, we have to take the shapes curvature into account.
(of the surface), (vectors)
On each point we have two directions which are perpendicular to each other.
Both are associated with a curvature (scalar value).
The first direction is passed to the pixel shader, as well as both curvatures.

Compute the view along the direction surface
direction ↳ we want the vector to be orthogonal (direction alongside the camera)

```
float3 viewDirection = normalize(input.posWorld.xyz - g_WorldSpaceCameraPos);

float3 normal = normalize(input.normal); (→ why normalize it) → because of the viewspace? not only → to get the angle) Mithilfe des Skalarprodukts kann sich der Winkel zw. 2 Vektoren bestimmen

float3 viewDirection_Surface = viewDirection - dot(normal, viewDirection) * normal;
```

Compute $\cos(\varphi)$, the dot product of the angle between view direction along the surface and the first curvatures direction.

```
float cos_phi = dot(normalize(viewDirection_Surface), normalize(curDir1));
```

Compute the curvature in view direction $c = c1 \cdot \cos^2(\varphi) + c2 \cdot \sin^2(\varphi)$
Don't call the sin function, exploit the fact that

$\cos^2 + \sin^2 = 1$

```
float sin = 1 - cos
float c = cos_phi_2 * c1 + c2 * sin_phi_2
float silhouetteFactor = abs(dot(viewDirection, normal));
if (silhouetteFactor < sqrt(c * (2 - c)))
{ color = float4(0, 0, 0, 1); }

return color;
```

view = pos - camera

Projection formel:

$g(\lambda) = v + \lambda \cdot n$

$\langle n, g(\lambda) \rangle = 0$

$\langle n, v + \lambda n \rangle = \langle n, v \rangle + \langle n, \lambda n \rangle = \langle n, v \rangle + \lambda \langle n, n \rangle = 0$

$\lambda \langle n, n \rangle = -\langle n, v \rangle$

$\lambda = -\frac{\langle n, v \rangle}{\langle n, n \rangle} = -\frac{\langle n, v \rangle}{\|n\|^2}$

Dot product: $a \cdot b = |a| \cdot |b| \cdot \cos(\theta)$
If both vectors are unit length, the dot product reduces to
$a \cdot b = \cos(\theta)$
if $a \cdot b > 0$ → angle between the vectors is less than 90°
$<$ → greater than 90°
$a \cdot b = 0$ → orthogonal

$\left(\frac{1}{2}\right)\left(\left(\frac{2}{3}\right)^{\text{normalize}} \cdot \sqrt{22}\right)$ $\sqrt{x^2+y^2+z^2}$

$\rightarrow \frac{1}{\sqrt{22}} \cdot \begin{pmatrix} 2 \\ 3 \\ 3 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{22}} \\ \frac{3}{\sqrt{22}} \\ \frac{3}{\sqrt{22}} \end{pmatrix}$

$\begin{pmatrix} \frac{2}{\sqrt{22}} \\ \frac{3}{\sqrt{22}} \\ \frac{3}{\sqrt{22}} \end{pmatrix} \cdot ($

Color-Blending:

$C_i = \alpha_i \cdot C_i + (1 - \alpha_i) * C_{i-1} \quad , i \in [1, N-1]$

result.rgb = src.rgb · src.a + dest.rgb · (1 - src.a)
result.a = src.a · 1 + dest.a · (1 - src.a)

Alpha blending creates a transparency effect in which the two colors are mixed based on the source alpha channel

We can consider the source alpha channel as a percentage slider.

Csource * SRC_ALPHA) + (Cdest * INV-SRC_ALPHA)

VS_OUTPUT vertex_shader(VS_INPUT IN, uniform bool fogEnabled)

VS_OUTPUT OUT = (VS_OUTPUT) 0;

OUT.Position = mul(IN.ObjectPosition, worldViewProjection);
OUT.TextureCoordinate

Phong Lighting formula: $2 \cdot (N \circ L) \cdot N - L$

$R = 2(N \circ L)N - L$ Specular Phong: $(V \cdot R)^S$

Normal Mapping: to fake the detail of a bumpy surface

Texture Mapping

To map a texture to a triangle, we include two-dimensional coordinates with each vertex. We use those coordinates to look up a color stored in a 2D texture.

And we do this lookup in our pixel shader to find the color for each pixel of the triangle.

Samplers controls how a color is retrieved from a texture.

float4 pixel_shader(VS_OUTPUT IN) SV_TARGET {
    return ColorTexture.Sample(ColorSampler, IN.TextureCoordinate);
}

sample() → ① argument: sampler object
② coordinates to look up in the texture

$\alpha = 1 \cdot src.a + (1 - src.a) \cdot dest.a$
$C = 1 \cdot src.a \cdot src.rgb + C_{dest}$

Shader Understanding

On the bottom of figure 2 the orientation of the coordinate axes can be seen.

```
struct Vertex
{
    float4 pos : SV_POSITION;
    float2 TextureCoordinate : TEXCOORD0;
};
```

`[maxvertexcount(3)]` → outputs 3 vertices

```
void gs( triangle Vertex input[3], inout TriangleStream<Vertex> Stream)
```

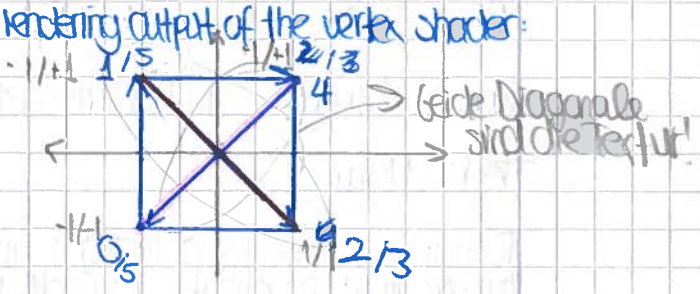
3 vertices per triangle

```
for (int i=0; i<3; i++)
{ Vertex v;
  v.pos = input[i].pos;

  if (i==1)
  { v.pos = float4(2,1,1,1); }
  v.pos = input mul(v.pos, g_worldViewProj);

  v.TextureCoordinate = input[i].TextureCoordinate;
  Stream.Append(v);
}
}
```

gets 3 vertices

v.pos = input[0].pos = (-1, -1, 0, 1);
v.pos = input[1].pos = (2, 1, 1, 1)
v.pos = input[2].pos = (1, -1, 0, 1)

rendering output of the vertex shader:



0 : (-1, -1, 0.0) (1.0)

1. (-1, 1, 0.0, 1.0)

2. (1, -1, 0.0, 1.0)

3. (1, -1, 0.0, 1.0)

4. (1, 1, 0.0, 1.0) (2, 1, 1, 0.1) 5: (-1, 1, 0, 1);

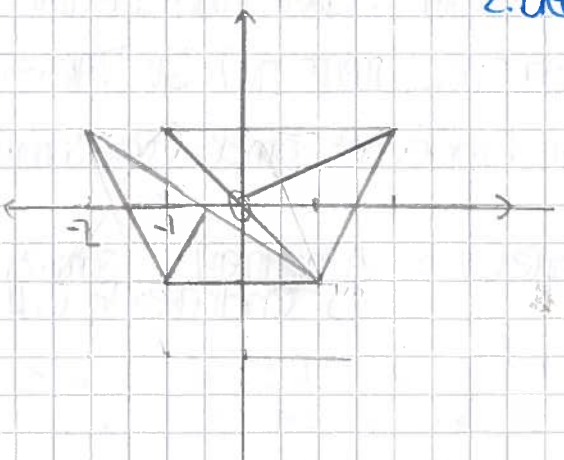
reduzierte drawing

Question I have: Wird für jeden einzelnen Punkt ein Dreieck erstellt?

(-1, 1) → (-2
↳ verdoppelt sich!!

1. Dreieck

2. Dreieck



(b) Draw the rendering output

float4 PS(Vertex input) : SV_Target

float2 tex = input.TextureCoordinate;

tex.y *= 0.5f; (tex.x += 0.0f) → alle TextureKoordinaten werden auf 0.5·y umgerechnet

float4 color = Texture0.SampleLevel(SamplerStateLinear, tex, 0); (Grundgerüst)

(c) Endterm

Training on writing a vertex shader:

- Should transform vertex positions to clip space ✓
- normals to view space (normalize)

void VertexShader(in VSIN input, out PSSIN output){

output.pos = mul(float4(input.pos.xyz, 1), g_WorldViewProj);

// normalizing not important here, because g_WorldViewProj already converts everything into clipspace

output.normal = normalize(mul(float4(normal.xyz, 0)), g_WorldViewMatrix);

// normalizing is important here because of the worldview normals which only converts everything to view space

Pixel Shader:

float4 PS(in PSIN frag): {

float4 color =

g_diffuse.Sample(sampler, frag.uv);

Reminder: Phong Lighting: (when considering diffuse, ambient etc.)

$$k_d \cdot (\vec{E} \cdot \vec{R}) + k_s \cdot (\vec{V} \cdot \vec{r})^n + k_a$$

Diffuse (Lambertian) reflection at rough material: $k_d \cdot (\vec{n} \cdot \vec{e})$

float L = dot(normalize(-frag.normal), g_LightDirection.xyz);

return color · L · g_LightColor;

Array filtering:

$$Color = \alpha_{src} \cdot color_{src} + (1 - \alpha_{src}) \cdot color_{dest}$$

$$BS_1: \quad 0.5 \cdot \begin{pmatrix} 0.4 \\ 0.4 \\ 0 \end{pmatrix} + (1 - 0.5) \begin{pmatrix} 0.4 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.4 \\ 0.2 \\ 0 \end{pmatrix}$$

α_{src} *2 | *2 | *1

$$\alpha_{Blend} = \alpha_{src} \cdot 1 + (1 - \alpha_{src}) \cdot \alpha_{dest}$$

$$= 0.5 + (1 - 0.5) \cdot 0.5$$

$$= 0.5 + 0.5 \cdot 0.5$$

$$= 1 \cdot 0.5 = 0.5 \quad 0.75$$

Motion Dynamics

(a) Projectile Motion:

$x(t_0) = (x_0, y_0)$ Pos the initial velocity v_0

Explicit Euler $y(t+h) = y(t) + h v_y(t, x(t))$

2014 Klausur 1 a)

- Gravitation does not act on the projectile
- Projectile moves with a velocity of $v(t) = t\ m/s$ into the direction $(1,0)$. t is the travel time of projectiles.

projectile travels over a period of 2 seconds, starting at travel time 10s using an Explicit Euler with a timestep $h = 1s$

$x(0) = 10$

After 1s: $x(11) = x(10) + 1 \cdot v(10) = 0 + 1 \cdot 10 = 10$

$x(12) = x(11) + 1 \cdot v(11) = 10 + 1 \cdot 11 = 21$

$v(10) = 10$ $\quad v(t) = t\ m/s$ → t is the travel time of projectiles

It starts with 10s → $t = 10$ → $v(10) = 10$ (because of the definition)

$v(11) = 11$ $\quad$ After 2s $x(12) = x(11) + h \cdot v(11)$

$= 10 + 1 \cdot 11 = 21$

(b) - $v = 10\ m/s$ in the direction $(1,0)$

- at some time t_0, acceleration of $10\ m/s^2$ starts acting on the projectile into the direction of movement.

20

$h = 1$ $\quad$ time frame: 2s $\quad$ using Semi-Explicit Euler:

$$v_y(t+h, x(t)) = v_y(t, x(t)) - hg$$

$$y(t+h) = y(t) + h v_y(t+h, x(t))$$

10.9.61

$v_0(t_0) = 10$ $\quad x(t_0) = 0$ $\quad$ After 1s: $v(t_0+1) = v(t_0) + 1 \cdot a(t_0+1)$

$x(t_0+1) = x(t_0) + 1 \cdot v(t_0+1)$

$v_{old} + h \cdot a = v_{new}$

$v_{new} = v_{old} + a \cdot h$ → $v(t_0+1) = v(t_0) + a(t_0+1)$

$p_{new} = p_{old} + v_{old} h$

$v_0(t_0) = 10$ $\quad x(t_0) = 0$

After 1s: $v(t_0+1) =$

Assigment 2

a. When using Phong illumination model and decreasing the angle between incoming light direction and the normal vector, the reflection at a point on a purely specular reflecting surface always appears brighter.

Wrong: The angle between R and V may decrease so that brightness decreases.

(The brightness of a glossy reflection decreases as the angle between the view direction and the ideal reflection direction increases.)

Reminder:
Specular: $(V \cdot R)^s$
Phong

b. Gouraud and Phong shading give the same result when the surface reflects only diffusely.

$R = 2 \cdot (N \cdot L) \cdot N - L$

Interpolation von Phong-Beleuchtung und Normalen liefert andere Ergebnisse als Phong-Beleuchtung und Farbwerten.

Specular reflection: (for smooth metallic or highly polished objects)

$I_s = k_s (\vec{r} \cdot \vec{v})^p I_s$

Assigment 3

Alpha-Blending

~~result.rgb = src.rgb · src.a + dest.rgb · src.a~~
~~result.a = src.a · BlendAlpha + dest.a · DestBlendAlpha~~

new color being written is called the **source color**

destination color is the color that already exists in the render target.

Phong shading:
Higher quality shading interpolating per-vertex normals and evaluating for every surface point the illumination model

Lambertian reflection:

$I_r = k_d \cdot (\vec{n} \cdot \vec{l})$

Gouraud shading:
compute reflected color per vertex and interpolate resulting color smoothly.

- druckenV
-

result.rgb = src.rgb · srcBlend + dest.rgb · destBlend

result.a = src.a · srcBlendAlpha + dest.a · destBlendAlpha

Depth testing: makes use of the distance between the object and the camera for each corresponding pixel written to the render target.

depth testing is enabled and set to "less":

$c = src.rgb \cdot (src.a) + dest.rgb \cdot (1 - src.a)$

$c = 0.5 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + (1 - 0.5) \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

depth testing is disabled: objects are rendered in the order of increasing index

$c_1 =$  $src.rgb = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \cdot dest.rgb = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ $src.a = 0.5$

Blend 1 and 2

$\rightarrow c = 0.5 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + (1 - 0.5) \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

Blend 2 and 3 $\quad src.rgb = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ $src.a = 0.5$

$0.5 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + (1 - 0.5) \cdot \left(0.5 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + (1 - 0.5) \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right)$

e.) back-to-front oder

$c = 0.5 \cdot \begin{pmatrix}1\\0\\0\end{pmatrix} + (1-0.5)\begin{pmatrix}0\\1\\0\end{pmatrix}$

src rgb = $\begin{pmatrix}1\\0\\0\end{pmatrix}$ src.a = 0.5

dst rgb = $\begin{pmatrix}0\\1\\0\end{pmatrix}$

3 / 1

back to front (farthest from the camera to nearest)

~~nearest = 3~~ ~~farthest: 1~~ → es stand in der Aufgabenstellung, dass von 1 zu 3 gerendert wird. Hier nimmt man einfach den Punkt, der am nächsten liegt sowie auch den Punkt, der am weitesten vom Pixel entfernt ist.

3. compute the value of the diffuse intensity

$I_d = k_d \cdot (\vec{n} \circ \vec{l}) = 0.5 \cdot \cos(0°)$

- Wenn man als Alpha-Wert 1 hat, dann ist man nicht transparent.
- wenn depth-test auf "less" gestellt wird, werden im Prinzip nur Objekte mit niedriger Tiefe zu anderen gerendert

→ wenn jetzt aber depth-test auf 2 higher gestellt wurde, dann werden die Obje

3. (Illumination)

a) compute the value of the diffuse reflected intensity.

$I_r(x, \omega_r) = k_d \cdot (\vec{n} \circ \vec{l}) \underbrace{I_i(x, \omega_i)}_{\hookrightarrow = 1} = 0.5 \cdot 1 \cdot \cos(0°) = 0.5$

↳ diffuse reflection is maximal when it is perpendicular to the light direction ($\vec{n} \circ \vec{l}$)

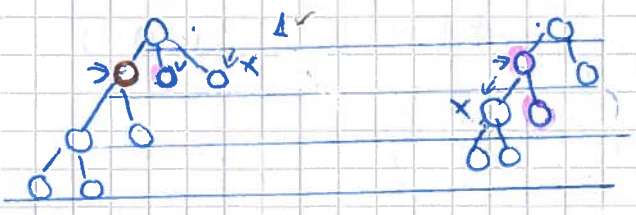
b) compute the value of the specular reflection (when its maximal).

maximum specular reflectance occurs when the viewpoint is along the path of the perfectly reflected ray.

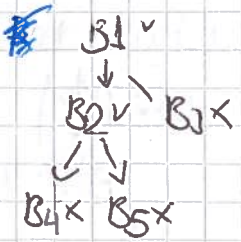
$I_r(x, \omega_v) = k_s (\vec{v} \cdot \vec{r})^n \cdot I_i(x, \omega_e)$

$= 0.5 (\vec{v} \circ \vec{r})^{\sqrt{2}} \cdot 1 = 0.5 \cdot \cos(0°)^{\sqrt{2}} \cdot 1 = 0.5$

Collision Detection:



1

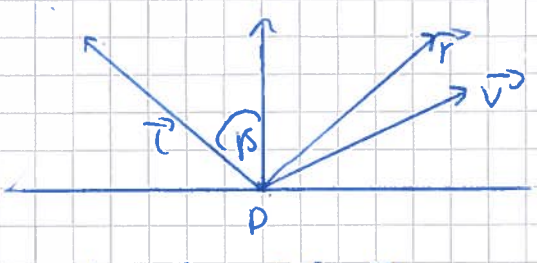


(c) 4×4 transformation matrix

cube's position → (2,2,2)

$$\begin{pmatrix}1&0&0&2\\0&1&0&2\\0&0&1&2\\0&0&0&1\end{pmatrix}\cdot\begin{pmatrix}5&0&0&0\\0&5&0&0\\0&0&5&0\\0&0&0&1\end{pmatrix}\cdot\begin{pmatrix}1&0&0&-2\\0&1&0&-2\\0&0&1&-2\\0&0&0&1\end{pmatrix}$$

(d)



should the $\vec{v}$-Vector always be orthogonal to the reflection / (light direction)

8.) ~~Collision~~ Testing

circle-circle-tests

(8,1), (8,2), (8,4,) (8,5)

3 tests?

g) y=0 Center of projection (0-6) → where's the minus?!

$$\begin{pmatrix}1&0&0\\0&1&-6\\0&0&1\end{pmatrix}\quad\begin{pmatrix}1&0&0\\0&0&0\\0&6&1\end{pmatrix}\rightarrow\begin{pmatrix}1&0&0\\0&0&0\\0&\frac{1}{6}&1\end{pmatrix}$$

(b) (100) after rotation (110)

cos 0 = 1
sin 1 0 = 0

$$\begin{pmatrix}1&0&0\\1&1&0\\0&0&1\end{pmatrix}$$

welche Matrix und den Ausgangspunkt gilt
A * P = P' (neuer Punkt)

(100) auf (110) mappen $\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}=\begin{pmatrix}1\\1\\0\end{pmatrix}$?

$$\begin{pmatrix}1&0&0\\1&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}=\begin{pmatrix}1\\1\\0\end{pmatrix}$$

$$\begin{pmatrix}1&-1&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}0\\1\\0\end{pmatrix}=\begin{pmatrix}-1\\+1\\0\end{pmatrix}$$

(010) → (-110) ?

↳ $\begin{pmatrix}1-10&0\\1+10&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$

Blending in DirectX

BlendState BS1

α-Blending : enabled

Blending: B = 3 / A = 2

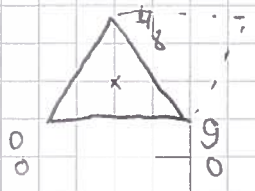
Blending: A = 1,2 (0.4 0.02 0.0) BS
B = (0.3, 0.3, 0.4)

BS: A = 2 (0.4 0 0)
B = 3 (0.4 0 0)

BlendEnable [ ] = TRUE;
SrcBlend[0] = ONE;

$\frac{1}{2}\sqrt{72^2 + (-72)^2}$ ... $= \sqrt{72^2 + (-72)^2} \cdot \frac{1}{2}$

4 9
2 0
4 9
2 0 $= 2 \cdot 9 - 4 \cdot 0$



$\alpha_2 = \frac{\frac{1}{2}|\vec{v_1 P} \times \vec{v_1 v_3}|}{\frac{1}{2}|\vec{v_1 v_2} \times \vec{v_1 v_3}|} = \frac{|\binom{4}{2} \times \binom{9}{0}| + |\binom{18}{-18}|}{|\binom{72}{-72}|}$

$\alpha_3 = \frac{\frac{1}{2}|\vec{v_1 P} \times \vec{v_1 v_2}|}{\frac{1}{2}\, mt} = \frac{\sqrt{18^2 + (-18)^2}}{\sqrt{72^2 + 72^2}} = \frac{\sqrt{648}}{\sqrt{72^2+72^2}} = \frac{18\sqrt{2}}{8 \cdot 9\sqrt{2}} = \frac{18}{72} = \frac{2}{8} = \frac{1}{4}$

$\vec{v_1 v_3} = \binom{0}{0} + \binom{-9}{0} = \binom{9}{0}$

$\alpha_3 = |\vec{v_1 P} \times |\vec{v_1 v_2}|$

$= \binom{4}{2}\binom{4}{8} = 2 \cdot 4 - 4 \cdot 8 = 8 - 32 = -$

4 4
2 8
4 4
2 8

$\vec{v_1 P} = P - v_1 = (4,0,7) - (1,2,7) = 3\,2,0$

$\vec{v_1 v_3} = (5,6,7) - (1,2,7)$

Projectile Motion

- compute the velocity $v = 2\,m/s$ $g = x$ air resistence: x 9,8

$S = t \cdot \ldots \Rightarrow$ $v = s/t$

$p_1(1) = \binom{10}{10} + 1 \cdot \binom{2}{0} = \binom{12}{10}$ op. v. ↳ influence of g.

$p_2(1) = \binom{12}{10} + 1 \binom{2}{0} - \binom{14}{10}$

(b) mass 2kg $v = \sqrt{2}\ m/s$ aber Vektor ist $= \binom{1}{1}$ → weil 45° $|\vec{v}| = \sqrt{2}\ m/s$

$a = \binom{0}{10} \cdot 2 = \binom{0}{-20}$

$v_{new} = v_{old} + 1 \cdot a$

$v_{new} = \binom{1}{1} + 1 \cdot \binom{0}{-20} = \binom{1}{-19}$

$p_{new} = \binom{10}{10} + 1 \cdot \binom{1}{1} = \binom{11}{11}$

$p_1(1) = \binom{11}{11}$

$v_{neu} = \begin{pmatrix} 1 \\ -9 \end{pmatrix}$ $\quad p = \begin{pmatrix} 10 \\ 10 \end{pmatrix} + 1 \cdot \begin{pmatrix} 1 \\ -9 \end{pmatrix} = \begin{pmatrix} 11 \\ 1 \end{pmatrix}$

Collision Detection:

BC
CE
EF

Überschneidung bei x- und y-Achse

$v = \frac{s}{t}$ $\quad \Rightarrow \quad t/s = 1/v \Rightarrow t = s/v \quad \Rightarrow s = v \cdot t$

t_1 = First contact

t_2 = last contact

$[2 \cdot t_1 = (100 - 4 - 6) - 8m/s \cdot t_1]$

$2t_1 = 90 - 8m/s \cdot t_1$

$10t_1 = 90$

$t_1 = 9s$

$2t_1 + 8t_1 = 8\ 90$

$vt + vt = s$

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ → man müsste ja keine Skalierung haben...

b) - → $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(-\alpha) & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\frac{1}{\sqrt{2}}(-1,1,0) + \frac{1}{\sqrt{2}}(1,1,0) = \frac{1}{\sqrt{2}}(0,2,0) : 2 = (0,1,0)$

7) Shading and Lightning

point light source $(-10,0,0)$ position p: $(0,0,0)$ normal: $(2,1,0)$

light direction: $(-10,0,0) - (0,0,0) = (-10,0,0)$

$$\vec{r} = 2(\vec{n} \circ \vec{e})\,\vec{n} - \vec{e}$$

$$\vec{n} \circ \vec{e} = \begin{pmatrix}-10\\0\\0\end{pmatrix} \circ \begin{pmatrix}2\\1\\0\end{pmatrix} = -20$$

$$2(-20)\,\vec{n} - \vec{e} = -40 \cdot \begin{pmatrix}2\\1\\0\end{pmatrix} - \begin{pmatrix}-10\\0\\0\end{pmatrix} = \begin{pmatrix}-80\\-40\\0\end{pmatrix} - \begin{pmatrix}-10\\0\\0\end{pmatrix}$$

$$= \begin{pmatrix}-70\\-40\\0\end{pmatrix}$$

Normale muss normiert sein → man muss diese mal aber das negative von der Normale nehmen, da man anhand einer Skizze sehen kann, dass die Ebene von unten beleuchtet wird

$\vec{e}$ nicht...

$(-2,-1,0)$ in normierter Form: $\sqrt{(-2)^2+(-1)^2+0^2} = \sqrt{5}$ $\quad \frac{1}{\sqrt{5}}\begin{pmatrix}2\\1\\0\end{pmatrix}$

$$\rightarrow \vec{r} = 2\left(\frac{1}{\sqrt{5}}\begin{pmatrix}2\\1\\0\end{pmatrix} \circ \begin{pmatrix}-10\\0\\0\end{pmatrix}\right)\cdot \vec{n} - e$$

$$= 2\left(\frac{1}{\sqrt{5}} \cdot 20\right) \cdot \begin{pmatrix}-2\\-1\\0\end{pmatrix} - \begin{pmatrix}10\\0\\0\end{pmatrix}$$

$\frac{40}{5} = 4 \cdot \sqrt{5} \cdot \frac{1}{\sqrt{5}}$

$$\vec{r} = \frac{40}{\sqrt{5}}\begin{pmatrix}-2\\-1\\0\end{pmatrix} - \begin{pmatrix}10\\0\\0\end{pmatrix}$$

$$\frac{-40}{\sqrt{5}}\begin{pmatrix}-2\\-1\\0\end{pmatrix} * \frac{1}{\sqrt{5}} = \frac{-40}{5}$$

$$= \frac{40}{5} \cdot 8\begin{pmatrix}-2\\-1\\0\end{pmatrix} = \begin{pmatrix}-16\\-8\\0\end{pmatrix} - \begin{pmatrix}10\\0\\80\end{pmatrix} = \begin{pmatrix}-26\\-8\\0\end{pmatrix} =$$

$$= \begin{pmatrix}-16\\-8\\0\end{pmatrix} + \begin{pmatrix}10\\0\\0\end{pmatrix} - \begin{matrix}-6\\-8\\0\end{matrix}$$

Klausur 2015 ad/h

Lightning and Shading

a) position $P = (3,1,-2)^T$ normale at this point $(0,1,0)$

$k_s = 0.6 \quad n = 8 \quad L_p = (1,3,-1) \quad C_L = (0.8, 1, 0.8)$

$E_p = (6,5,-2)$

$I_r(x_r, \omega_r) = k_s (\vec{r} \cdot \vec{v})^n \cdot C_L \cdot 0.6 \cdot C$

$\vec{r} = 2(\vec{n} \cdot \vec{l}) \cdot \vec{n} - \vec{l} = 2(3) \cdot \begin{pmatrix}0\\1\\0\end{pmatrix} - \begin{pmatrix}1\\3\\-1\end{pmatrix}$

$= (2,2,-1)$?

$= 3 \cdot 2 = 6$

$\vec{l} = (1,3,-1) \rightarrow \sqrt{1+3^2+(-1)^2} = \sqrt{11}$

$\vec{l} = \frac{1}{\sqrt{11}} \begin{pmatrix}1\\3\\-1\end{pmatrix}$ normieren

$6 \cdot \begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}0\\6\\0\end{pmatrix}$

$\vec{n} = (0,1,0) \rightarrow \sqrt{0+1^2+0^2} = 1$ normiert n $\rightarrow (0,1,0)$

$\begin{pmatrix}0\\6\\0\end{pmatrix} - \begin{pmatrix}1\\3\\-1\end{pmatrix} = \begin{pmatrix}-1\\3\\+1\end{pmatrix}$

$\frac{1}{\sqrt{11}} \begin{pmatrix}1\\3\\-1\end{pmatrix} \begin{pmatrix}0\\1\\0\end{pmatrix} = \frac{1}{\sqrt{11}} \begin{pmatrix}0\\3\\0\end{pmatrix}$?

p liegt nicht 000 $\rightarrow$

$L_p + p \cdot (1,3,-1) - (3,1,-2) = (-2, 2, +\frac{1}{8})$ $(\vec{e})$

~~$\sqrt{(-2)^2+2^2+(\frac{1}{8})^2} = \sqrt{9}$~~ $\sqrt{(-2)^2+2^2+1^2} = \sqrt{5}$ = ~~$\frac{1}{8}$~~

$2\sqrt{5}\,(\vec{n} \cdot \vec{e}) = \begin{pmatrix}0\\1\\0\end{pmatrix} \circ \begin{pmatrix}-2\\2\\1\end{pmatrix} = 2$ $\quad \begin{pmatrix}-2/3\\2/3\\1/3\end{pmatrix} \circ \begin{pmatrix}0\\1\\0\end{pmatrix}$

$\hookrightarrow 2\sqrt{5} \cdot 2 = 4\sqrt{5} \cdot \begin{pmatrix}0\\1\\0\end{pmatrix} - ($

$4 \cdot \begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}0\\4\\0\end{pmatrix} - \begin{pmatrix}1\\-3\\-1\end{pmatrix} =$

$= \frac{4}{3}\begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}0\\4/3\\0\end{pmatrix} - \begin{pmatrix}-2/3\\2/3\\1/3\end{pmatrix} = \begin{matrix}2/3\\4/3\\1\end{matrix}$ $\quad \begin{matrix}2/3\\2/3\\1/3\end{matrix}$

complete bullshit ...

$\frac{4}{3} - 2 \quad (3)$

$= \frac{4}{3} - \frac{6}{3} = -\frac{2}{3}$

$\rightsquigarrow \frac{1}{3}\begin{pmatrix}+2\\4/3\\-1\end{pmatrix}$

um v herauszukriegen: $E_p - p$ und dann normieren!

$(6,5,-2) - (3,1,-2) = (3,4,0)$ $\quad \sqrt{3^2+4^2+0} = \sqrt{9+16}$

$= \sqrt{25} = 5$

$\hookrightarrow (\frac{3}{5}, \frac{4}{5}, 0)$

7) Shading and Lightning

point light source $(-10,0,0)$ Position $p: (0,0,0)$ normal: $(2,1,0)$

Light direction: $(-10,0,0) - (0,0,0) = (-10,0,0)$

$\vec{r} = 2(\vec{n} \circ \vec{e})\,\vec{n} - \vec{e}$

$\vec{n} \circ \vec{e} = \begin{pmatrix} -10 \\ 0 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = -20$

$2(-20)\,\vec{n} - \vec{e} = -40 \cdot \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} -10 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -80 \\ -40 \\ 0 \end{pmatrix} - \begin{pmatrix} -10 \\ 0 \\ 0 \end{pmatrix}$

$= \begin{pmatrix} -70 \\ -40 \\ 0 \end{pmatrix}$

Normale muss normiert sein → man muss diese mal aber das negative von der Normale nehmen, da man anhand einer Skizze sehen kann, dass die Ebene von unten beleuchtet wird

e nicht...

$(-2,-1,0)$ in normierter Form: $\sqrt{(-2)^2 + (-1)^2 + 0^2} = \sqrt{5}$ $\quad \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$

$\rightarrow \vec{r} = 2\left(\frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}\right) \cdot \vec{n} - e$

$= 2\left(\frac{1}{\sqrt{5}} \cdot 20\right) \cdot \begin{pmatrix} -2 \\ -1 \\ 0 \end{pmatrix} - \begin{pmatrix} 10 \\ 0 \\ 0 \end{pmatrix}$

$\frac{40}{5} = 4 \cdot \sqrt{5} \cdot \frac{1}{\sqrt{5}}$

$\vec{r} = \frac{40}{\sqrt{5}} \begin{pmatrix} -2 \\ -1 \\ 0 \end{pmatrix} - \begin{pmatrix} 10 \\ 0 \\ 0 \end{pmatrix}$

$\frac{-40}{\sqrt{5}} \begin{pmatrix} -2 \\ -1 \\ 0 \end{pmatrix} * \frac{1}{\sqrt{5}} = \frac{-40}{5}$

$= \frac{40}{5} \cdot 8 \begin{pmatrix} -2 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} -16 \\ -8 \\ 0 \end{pmatrix} - \begin{pmatrix} 10 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -26 \\ -8 \\ 0 \end{pmatrix} =$

$= \begin{pmatrix} -16 \\ -8 \\ 0 \end{pmatrix} + \begin{pmatrix} 10 \\ 0 \\ 0 \end{pmatrix} - \begin{matrix} -6 \\ -8 \\ 0 \end{matrix}$

Formel für gleichmäßig beschleunigte Bewegung (Geschwindigkeit-Zeit-Gesetz):

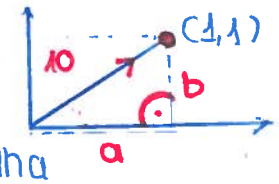
$v = a \cdot t + v_0$ v_0 ist die Anfangsgeschwindigkeit in Meter pro Sekunde

Bei Position nach Zeitabständen gilt: $x(t+h) = x(t) + h \cdot v_x(t+h, x(t))$

Motion Dynamics

projectile starts at t_0 with $v(t_0) = 10\ m/s$ into the direction $(1,1)$.
acceleration: $1\ m/s^2$ into the direction $(-1,0)$.

a) Computation of the vertical and horizontal component of the initial velocity



wegen (1/1) gilt immer, dass a=b!

$a^2 + b^2 = 10$
$2a^2 = 10^2 \mid :2$
$a^2 = \frac{10^2}{2} \mid \sqrt{}$
$v_{0x}: a = \frac{10}{\sqrt{2}}$ $b: \frac{10}{\sqrt{2}} = v_{0y}$

b) using explicit Euler scheme with a time step of $h = 1\ s$ we should compute the velocity after 1s

$v_{new} = v_{old} + a \cdot h$
initial velocity: $v(t_0) = 10\ m/s$
$\rightarrow v(t_0 + 1) = v(t_0) + a(t_0 + 1) \cdot h$
$= 10\ m/s + 1\ m/s^2 \cdot 1\ s = 11\ m/s$

Explicit Euler:
$y(t+h) = y(t) + h \cdot v_y(t, x(t))$

horizont $v_R = v_h + h \cdot 1\ m/s = \left(\frac{10}{\sqrt{2}} - 1\right) m/s$
vertic $v_V = v_V + h \cdot g = \left(\frac{10}{\sqrt{2}} - 10\right) m/s$

~~da ja nur nach der Best~~ Geschwindigkeit gefragt wurde wenn man diese auf die normale Art und Weise updaten:
$v_{new} = v_{old} + h \cdot a$ allerdings müssen wir ebenfalls beachten, dass diese Geschwindigkeit sowohl eine vertikale als auch eine horizontale Komponente hat:

$v_{horizontal} = v_{old} + a \cdot h = \left(\frac{10}{\sqrt{2}} - 1\right) m/s$

$v_{vertical} = v_{old} + g \cdot h = \left(\frac{10}{\sqrt{2}} - 10\right) m/s$

c) compute position of particle 1s after t_0 with $1\ s = h$

~~$v_{R_1} = v_R + h \cdot 1$~~ $v_{1,new}$ $\left(\frac{10}{\sqrt{2}} - 10\right) m/s$

compute position relative to its starting position after 1s
-> this means we need to calculate with (0/0) first (which is the starting position)

$v_{new} = v_{old} + g \cdot h = 0 + \left(\frac{10}{\sqrt{2}} - 10\right) m/s + \left(\frac{10}{\sqrt{2}} - 10\right) m/s$

$v_{h\,new} = v_{h\,old} + a \cdot h = 0 + \left(\frac{10}{\sqrt{2}} - 1\right) m/s = \left(\frac{10}{\sqrt{2}} - 1\right) m/s$

(c) Mirroring at the plane $z = x$

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

7. c)
$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{\sqrt{2}} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{pmatrix}$$

① Rotation um die z-Achse (45°)
② Rotation um die x-Achse (90°)
③ Rotation um die z-Achse (−45°)

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

Collision detection:

Sphere 1: $S_1(t) = S_1(0) + t \cdot v_1$
Sphere 2: $S_2(t) = S_2(0) + t \cdot v_2$

Distance between spheres at time t: $[S_1(t) - S_2(t)]^T \cdot [S_1(t) - S_2(t)]$

At time of contact it must hold:

$[S_1(t) - S_2(t)]^T \cdot [S_1(t) - S_2(t)] = (r_1 + r_2)^2$

Barycentric Interpolation:

$P = \alpha_1 \cdot P_1 + \alpha_2 \cdot P_2 + \alpha_3 \cdot P_3 \quad (\alpha_1 + \alpha_2 + \alpha_3 = 1)$

$C = \alpha_1 C_1 + \alpha_2 \cdot C_2 + \alpha_3 C_3$
$\alpha_1 = \Delta P P_2 P_3 \,/\, \Delta P_1 P_2 P_3$

Blending

$C = \alpha_1 \cdot C_1 + (1 - \alpha_1) \cdot \alpha_2 \cdot C_2 \quad ; \quad C_D = C_D + (1 - \alpha_D) \cdot \alpha_S \cdot C_S$

$\alpha = \alpha_1 + (1 - \alpha_1) \cdot \alpha_2 \quad ; \quad \alpha_D = \alpha_D + (1 - \alpha_D) \cdot \alpha_{DS}$

Rasterization

- kann konfiguriert werden (Back face culling, view frustum culling)
- dividiert die Vertices durch w (so werden x und y auf der Viewplain berechnet)
- berechnet für jedes Dreieck die Pixel, durch welche sie gehen werden.
- per-Vertex Attribute werden für jedes Fragment interpoliert

Angle in degrees

	sin	cos
0	0	1
30	$1/2$	$\sqrt{3}/2$
45	$1/\sqrt{2}$	$1/\sqrt{2}$
60	$\sqrt{3}/2$	$1/2$
90	1	0

reminder:
first translate, then scale, then rotate

Perspective projections

standard projection onto the $z=1$ plane

First: bring z component into 4th component of result vector
Second: divide by z 4th component.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow x,y$$

1. $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ → weil $y=0$

2. $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1/6 & 1 \end{pmatrix}$

Vector $(1,0,0)$ should be aligned with the vector $(1,1,0)$

① $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$

Vector $(0,1,0)$ should be aligned with the vector $(-1,1,0)$

② $\begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$

① & ② : $\begin{pmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ ...

Mirroring at some point:

Example: $(4,0,0)$

$$\begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2013. 8)

~~8~~ 8,1 8,2 8,3 8,4 8,5 8,6 8,7

wenn 8,3 selbst wieder Kästen würden wir nicht mehr den Branch von 8,3 runtergehen, weil ja mit 8,3 keine Kollision stattfindet! Aber allein, dass man am Anfang schon auf Kollision mit 8,3 getestet hat war das bereits ein Grund es mitzuzählen!

triangle-triangle-Test: 3 (8,4) → ist viel einfacher, man muss nur schauen mit welchen Kreisen es noch kollidiert, und wenn ja dann schauen ob sie selbst Dreiecke hat.

Mirroring at the point (4,0,0)

1. move everything 4 units left so that the mirroring point is on main axis
2. do the mirroring
3. move everything 4 units right.

b2) translation about (-4,-4,-1) rotation about z
translation about (4,4,1)

→ first we need to translate everything in the opposite direction
(when asked, that we should execute a certain operation through a point)

Shear: $Sh = \begin{bmatrix} 1 & sh^y_x & sh^z_x & 0 \\ sh^x_y & 1 & sh^z_y & 0 \\ & & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ Shear parallel to y-axis has $x' = x$ and $y' = y + xx$

Shear Matrix

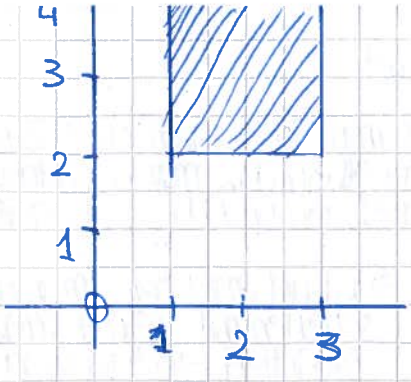
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ N_2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1/2 & 1 & 0 & 3/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

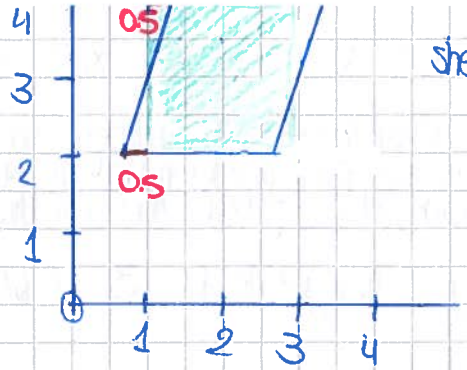
∧

Translation Matrix?

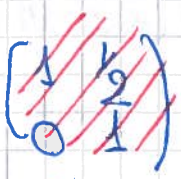
yes: $\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$

command — vector which will be moved with the command's data





Shear in the x-direction

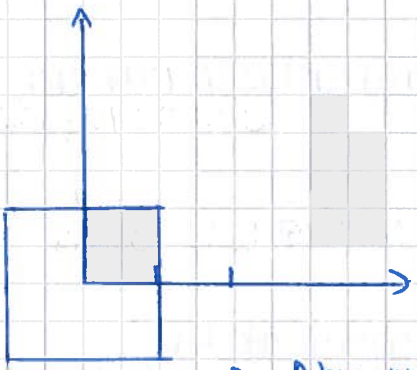


Shear parallel to the x axis: $x' = x + \lambda y$ and $y' = y$

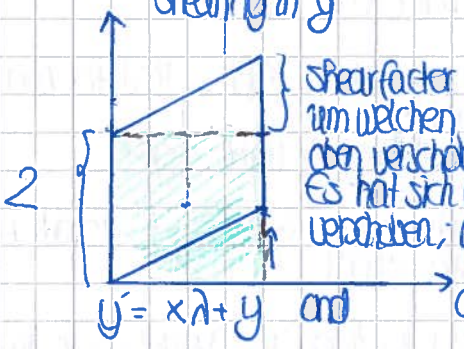
→ entlang der x-Achse wurden ja die Koordinaten verschoben (kommt in die x-Spalte)

$\begin{pmatrix} 1 & 1/2 \\ 0 & 1 \end{pmatrix}$

einfach schauen welche Achse "verschrägt" wurde, und wie um welchen Faktor die Koordinaten der "verschrägten" Achse verschoben wurde (und in welche Richtung)



Shearing in y



Shearfactor ½ → um welchen Faktor hat es sich nach oben verschoben? Es hat sich um eine Einheit nach oben verschoben, die ursprüngliche y-Koordinate lautete 2

$y' = x\lambda + y$ and also: $2 = 1 \cdot s$, $s = 1/2$

→ x-Achse wurde "verschrägt"
→ entlang der y-Achse werden die Koordinaten verschoben (kommt in die y-Spalte)

X

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Mine

$\begin{pmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Shearing in y with factor 0.5

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

distance moved = Shearfactor · distance from the invariant line

Invariant line = line which remains unchanged a certain

Shearing in x: a point moves by distance = its distance from x-axis

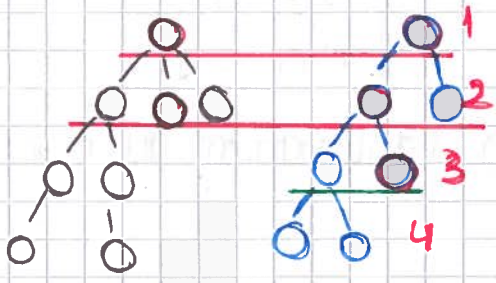
matrix: $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ → Shearing factor in x-direction

Shearfactor

~~Assignment 2~~ ~~The Big Model~~

$3 \cdot 2 = 6 + 2 + 2 = 10$

zweiten Hierarchiepunkt?



1. Test + 6 Tests + 12 Test = 8 Tests insgesamt

man achtet hier nicht darauf (im 1 Schritt ob sie kollidieren, sondern zählt einfach ob die Anzahl der Tests (Quaderanzahl) ab, die durchzuführen sind:

$n \cdot m \rightarrow n = 3 \quad m = 2 \Rightarrow 2 \cdot 3 = 6$ Tests

erst im zweiten Schritt schauen wir genauer nach ob zwischen den 3 Objekten überhaupt eine Kollision stattfindet oder nicht.

wenn ja: gehe eine Ebene vom Hierarchiebaum runter

wenn nein: ignorieren!

THM Rest 2

$$T^{-1} = \left(\begin{array}{cccc|cccc} \frac{1}{s} & 0 & 0 & -t_x & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & -t_y & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{s} & -t_z & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

$$+ \begin{array}{cccc|cccc} 1 & 0 & 0 & t_x & \frac{1}{s} & 0 & 0 & 0 \\ 0 & 1 & 0 & t_y & 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & 1 & t_z & 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}$$

$$= \begin{array}{cccc|cccc} 1 & 0 & 0 & t_x & \frac{1}{s} & 0 & 0 & -t_x \\ 0 & 1 & 0 & t_y & 0 & \frac{1}{s} & 0 & -t_y \\ 0 & 0 & 1 & t_z & 0 & 0 & \frac{1}{s} & -t_z \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}$$

THM Rest 2

$$T = \left(\begin{array}{cccc|cccc} s & 0 & 0 & s \cdot t_x & 1 & 0 & 0 & 0 \\ 0 & s & 0 & s \cdot t_y & 0 & 1 & 0 & 0 \\ 0 & 0 & s & s \cdot t_z & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

Folg 34

$$: s \rightsquigarrow \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & t_x & \frac{1}{s} & 0 & 0 & 0 \\ 0 & 1 & 0 & t_y & 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & 1 & t_z & 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & \frac{1}{s} & 0 & 0 & 0 & \frac{1}{s} \end{array}\right) \quad \rightsquigarrow \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & t_x & \frac{1}{s} & 0 & 0 & 0 \\ 0 & 1 & 0 & t_y & 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & 1 & t_z & 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{s} \end{array}\right) \cdot s$$

$$\Rightarrow \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & -t_x & \frac{1}{s} & 0 & 0 & 0 \\ 0 & 1 & 0 & -t_y & 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & 1 & -t_z & 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & \frac{1}{s} & 0 & 0 & -t_x \\ 0 & 1 & 0 & 0 & 0 & \frac{1}{s} & 0 & -t_y \\ 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{s} & -t_z \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

Assignment 5 (Transformation)

Rotation with matrices

Rotation mit x: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos & -\sin \\ 0 & \sin & \cos \end{bmatrix}$ mit y: $\begin{bmatrix} \cos & 0 & \sin \\ 0 & 1 & 0 \\ -\sin & 0 & \cos \end{bmatrix}$

mit z $\begin{bmatrix} \cos & -\sin & 0 \\ \sin & \cos & 0 \\ 0 & 0 & 1 \end{bmatrix}$ nur bei y ist das Vorzeichen beim unteren sin!

Difference between Phong and Gouraud shading in a situation where a specular highlight lies entirely in the interior of the triangle.

Gouraud shading: compute reflected color per vertex and interpolate resulting color smoothly → highlight can be missed if there is no vertex in highlight.

Phong shading: interpolating per-vertex normals and evaluating for every surface point the illumination model

Determining surface normals:

cross product of two sides of the triangle equals the surface normal.
Surface normal for a triangle can be calculated by taking the vector cross product of two edges of that triangle.

For triangle p1, p2, p3, if the vector U = p2 − p1 and the vector V = p3 − p1 then the normal N = U × V

2014 Transformations:

(b) $\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(90°) & -\sin(90°) & 0 & 0 \\ \sin(90°) & \cos(90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

(c) $\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

4.) Shading and Lighting.

Point light source (1, 10, 1) → reflects only diffusely viewer: (4, 6, 4)

highest intensity, when $(\vec{n} \circ \vec{L}) = 0$

$\hat{n}$ normal vector of the surface

7.a) specular reflecting plane

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = 0$$

point light: (10,0,0) compute the direction of the reflected light at the point in the plane at position (0,0,0)

formula: $2 \cdot (\vec{n} \circ \vec{e}) \vec{n} - \vec{e}$

~~Reflection normal~~: light direction: (-10,0,0) - (0,0,0) = (-10,0,0)

~~$\vec{n} \circ \vec{e} = \begin{pmatrix} -10 \\ 0 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} = -20$~~ ~~$\rightarrow 2 \cdot (-20) \cdot \vec{n} - \vec{e}$~~

~~$= -40 \cdot \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$~~

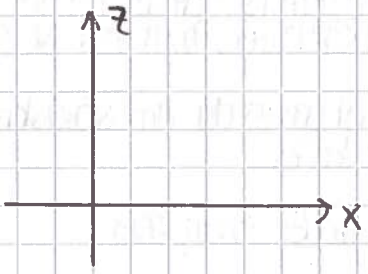
$\vec{r} = 2 \cdot (\vec{n} \circ \vec{e}) \vec{n} - \vec{e}$ $\vec{n} = (-2, -1, 0)^T$ normiert: $\frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$

$$\vec{r} = 2 \left(\frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 10 \\ 0 \\ 0 \end{pmatrix} \right) \vec{n} - \vec{e} = \frac{40}{\sqrt{5}} \begin{pmatrix} -2 \\ -1 \\ 0 \end{pmatrix} - \begin{pmatrix} 10 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -6 \\ -8 \\ 0 \end{pmatrix}$$

The plane $y = 0$ is called the (x,z) plane

$\vec{lp} = (1, 10, 1)$ viewer position = (4,6,4)

$\vec{e} = \vec{lp} - \vec{p} = (1, 10, 1) - (4, 6, 4)$
$= (-3, 4, -3)$



Klausur 2013 wdh:

$\vec{e} = \vec{lp} - \vec{p} = (1, 3, 1) \ominus (3, 1, 2) = (-2, 2, 1) \cdot (-2, 2, 1)$

~~$\vec{v}$~~ $= \vec{ep} - \vec{p} = (6, 5, -2) - (3, 1, -2) = (3, 4, 0)$

compute specular reflection at P

$\vec{n} = (0, 1, 0)$ Specular = $(R \cdot V)^n$ ~~Reflection~~: $w \cdot (\vec{v} \cdot \vec{r})^n$
Reflection: $2(\vec{n} \circ \vec{e}) \vec{n} - \vec{e}$

$\vec{n} \circ \vec{e} = \begin{pmatrix} -2 \\ 2 \\ 1 \end{pmatrix} \circ \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 2$ $\rightsquigarrow 2 \cdot 2 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} -2 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix} - \begin{pmatrix} -2 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$

Specular: $0.6 \cdot \left[\begin{pmatrix} 3 \\ 4 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} \right]^8 = 0.6 \cdot 14$

$= 3 \cdot 2 + 4 \cdot 2 + 0 \cdot 1$
$= 6 + 8 \cdot 14$

4.) Lighting $y=0$ $p=(1,10,1)-(0,0,0)\ (1,10,1)$

normalvector of y = $(0/1/0)$! $\vec{n}\circ\vec{e} = \begin{pmatrix}0\\1\\0\end{pmatrix}\circ\begin{pmatrix}1\\10\\1\end{pmatrix} = 0+10+0 = 10$

$\Rightarrow r = 2\cdot(\vec{n}\circ\vec{e})\vec{n}\cdot\vec{e} = 2\cdot 10\cdot\begin{pmatrix}0\\1\\0\end{pmatrix}-\begin{pmatrix}1\\10\\1\end{pmatrix} = 20\cdot\begin{pmatrix}0\\1\\0\end{pmatrix}\cdot\begin{pmatrix}1\\10\\1\end{pmatrix} = \begin{pmatrix}0\\20\\0\end{pmatrix}-\begin{pmatrix}1\\10\\1\end{pmatrix}$

$= \begin{pmatrix}-1\\10\\-1\end{pmatrix} = -\begin{pmatrix}1\\10\\1\end{pmatrix}$

a) which point does the viewer perceive the highest reflection?

$\vec{n}\circ\vec{e} = 0 \Rightarrow$ if this condition is valid $\quad p=(1,10,1)-(1,0,1)=(0,10,0)$

$\vec{n} = (0\ 1\ 0)$ $\quad ep=(4\ 6,4)-(1,0,1)=(3\ 6,3)$

All that matters is where the reflection is relative to the point light.
The highest concentration of intensity will be shined along the planes normal vector

Das diffuse Licht ist genau da am stärksten wo der Winkel zwischen dem Normalenvektor und des Lichtvektors minimal ist (wenn es also 0 ist! das Skalarprodukt)

specular ist genau da am stärksten wo der Winkel zwischen Viewer und Lichtquelle am geringsten ist.

↳ woher weißt man, etc.

$k_d\cdot(\vec{n}\circ\vec{e})$

$\cos(\dots) = 1$ wenn der Winkel 0° ist

Wir haben nur da eine Reflection wo der $\cos > 0$ ist

Winkel zw. reflektierten Licht und dem Viewvektor.
Wann scheint es genau auf unseren Blickwinkel?

↳ $(\vec{n}\circ\vec{e})$.

Skalarprodukt $\neq 0$
$\cos(0°) = 1$

- The scalar product between two normalised vectors yields the cosine of the angle between them

Skalarprodukt = 0 → wenn Vektoren 90° sind → $\cos(90°)=0$

an der Stelle an der n und l gleich sind - höchste diffuse Reflektion
an der Stelle an der v und r gleich sind. = höchste specular reflection

am höchsten wenn Skalarprodukt 1 ist → Winkel ist 0° & wenn Skalarprodukt 1 ist — $\cos(0)=1$

9) Transformation

(a) 2D perspective projection

Matrix representation of the standard projection onto the $z=1$ plane

first: bring z component into 4th component of the result vector

second: divide by 4th component

$z=-1$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \xrightarrow{\text{transform}} \begin{bmatrix} x^* \\ y^* \\ -1 \end{bmatrix} = p^*$$

homogenous vector

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix}$$

rescaling the vector by 1 over w

$z : (-1) = -z$

$-z \cdot (-1) = z$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \cdot \begin{bmatrix} x/z \\ y/z \\ -1 \end{bmatrix} \begin{matrix} x \\ y \\ z \end{matrix}$$

$\Rightarrow$ 

which value should w get so that after the homogenous division z will be -1?

$\frac{z}{w} = -1$ $\quad -z = z$ $\quad w = -z$

$\frac{1}{w} = -\frac{1}{z}$

$\rightarrow w = \frac{z}{-1}$

$w = -z$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -z \end{bmatrix}$$

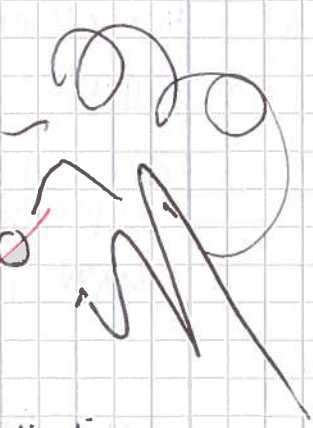
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \end{bmatrix}$$

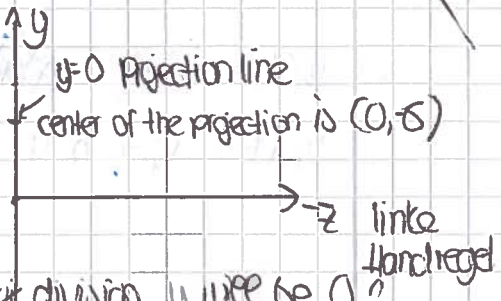
PERSPECTIVE MATRIX

onto the line $y=0$

$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} x/w & & \\ & 0 & \\ & & z/w \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$\frac{y}{w} = 0$





$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} \end{bmatrix}$$

which value should w get so that after the hom. clear division y will be 0?

$\frac{y}{w} = 0$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{6} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \cdot 1 \\ 0 \\ \frac{1}{6}+1 \end{bmatrix} = \begin{bmatrix} x \\ 0 \\ \frac{1}{6}+1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x/w \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{6} & 1 \end{bmatrix}$$

$z=1$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} - \begin{bmatrix} x/z \\ y/z \\ 1 \\ \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix}$$

$z/w = 1 \Rightarrow w = z$ da $z/z = 1$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \text{ div z projection } = \begin{bmatrix} x/z \\ y/z \\ 1 \\ \end{bmatrix}$$

(b) vector $(1,0,0)$ is aligned with $(1,1,0)$

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}$$

Projectile Motion:

a) Explicite: $y(t+h) = y(t) + h \cdot v(t, x(t))$

initial position = $(10/10)$
velocity: $2\,m/s$

$$p(11/10) = p(10/10) + 1 \cdot \begin{pmatrix} 2\,m/s \\ 0 \end{pmatrix} = \begin{pmatrix} 12 \\ 10 \end{pmatrix}$$

↳ no influence of gravity

b) $v_0(t_0) = \sqrt{2}\,m/s$

$v_0(t_0+1) = v_{t_0} + h \cdot a$

$$\begin{pmatrix} 1/2 & 1/2 & 1/\sqrt{2} \\ 1/2 & 1/2 & -1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \end{pmatrix} \begin{matrix} \cos(90°) & -\sin(90°) \\ \sin(90°) & \cos(90°) \end{matrix} = \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}$$