3D-Programmierung in C#

Über mich

Roland König

E-Mail: roland.koenig@rolandk.de

Blog: www.rolandk.de/wp

Berufliches

Schwerpunkt: Produktentwicklung, Innovationen

Arbeitgeber: IGZ Logistics + IT, Falkenberg

IGZ DIE SAP® INGENIEURE

3D-Programmierung in C#

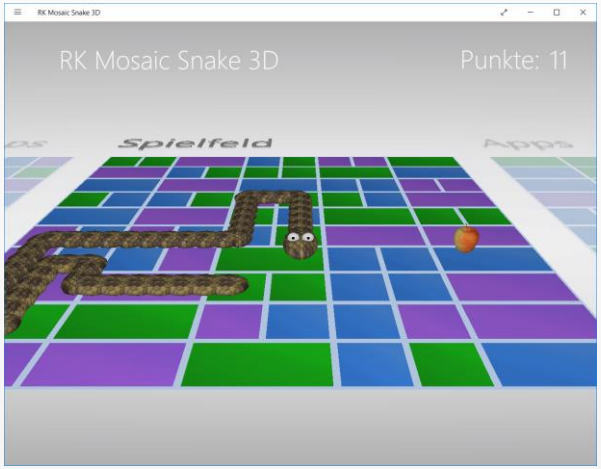
Agenda

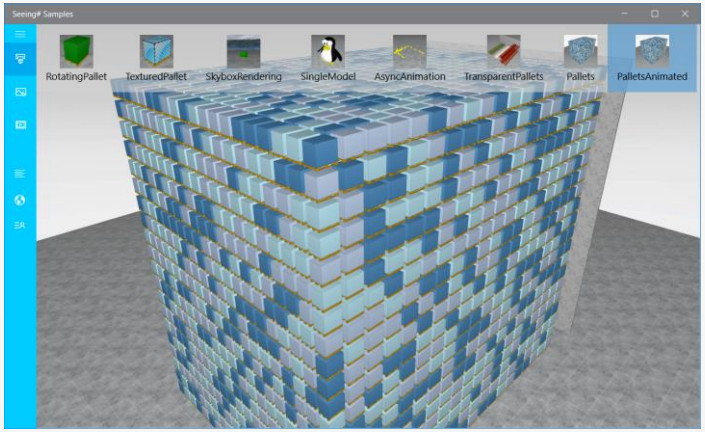
- **Übersicht Seeing#**
- Diverse Aspekte der Entwicklung
  - Multiplatform
  - Parallelisierung
  - Qualitätssicherung
  - 3D-Rendering
- Einstieg

Übersicht Seeing#

Was ist Seeing#?

- 3D/2D-Engine für C#

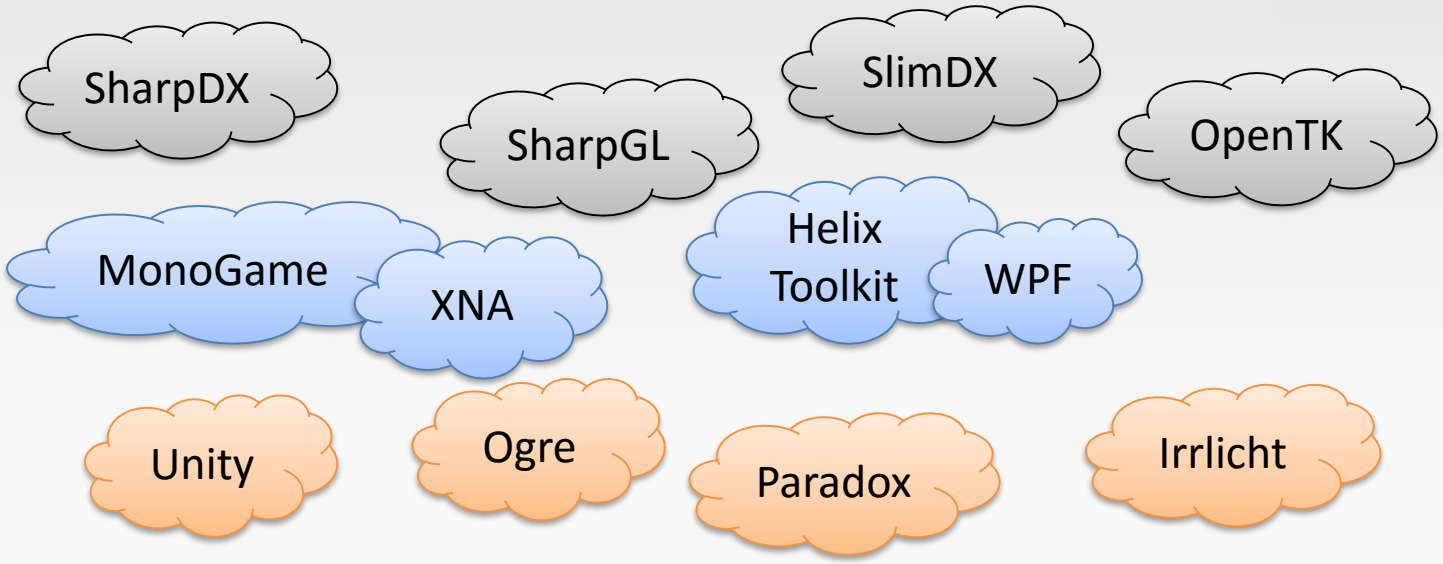




Übersicht Seeing#

Warum eine eigene 3D-Engine?

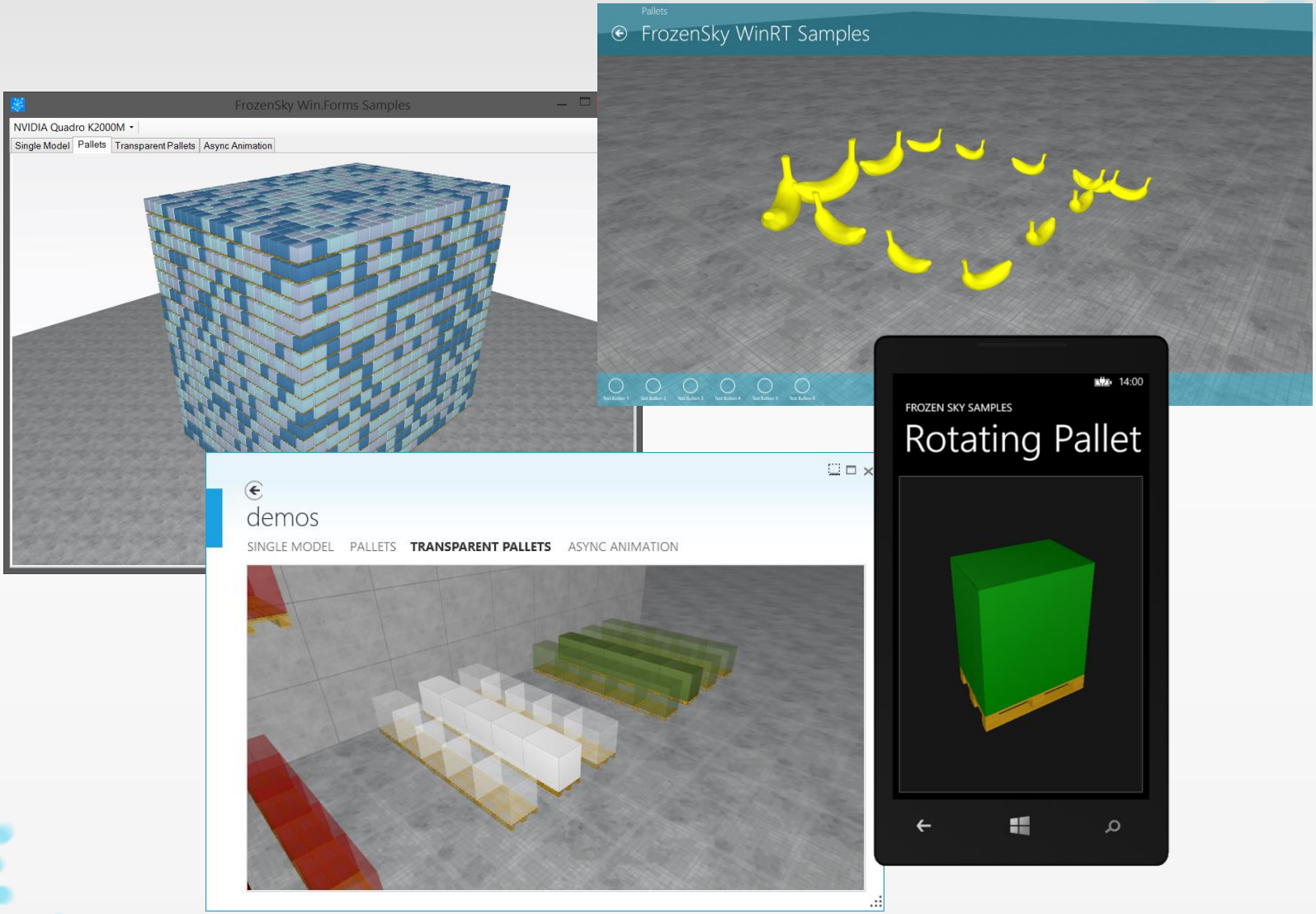
- Alternativen für 3D-Rendering in C#



- Gründe für Entwicklung Seeing#
  - Anderer Fokus
  - Remote Desktop sehr wichtig, Software-Rendering
  - Mehrere Views
  - Viele, per Coding definierte 3D-Objekte

Übersicht Seeing#

Herausforderung: Multiplatform



Übersicht Seeing#

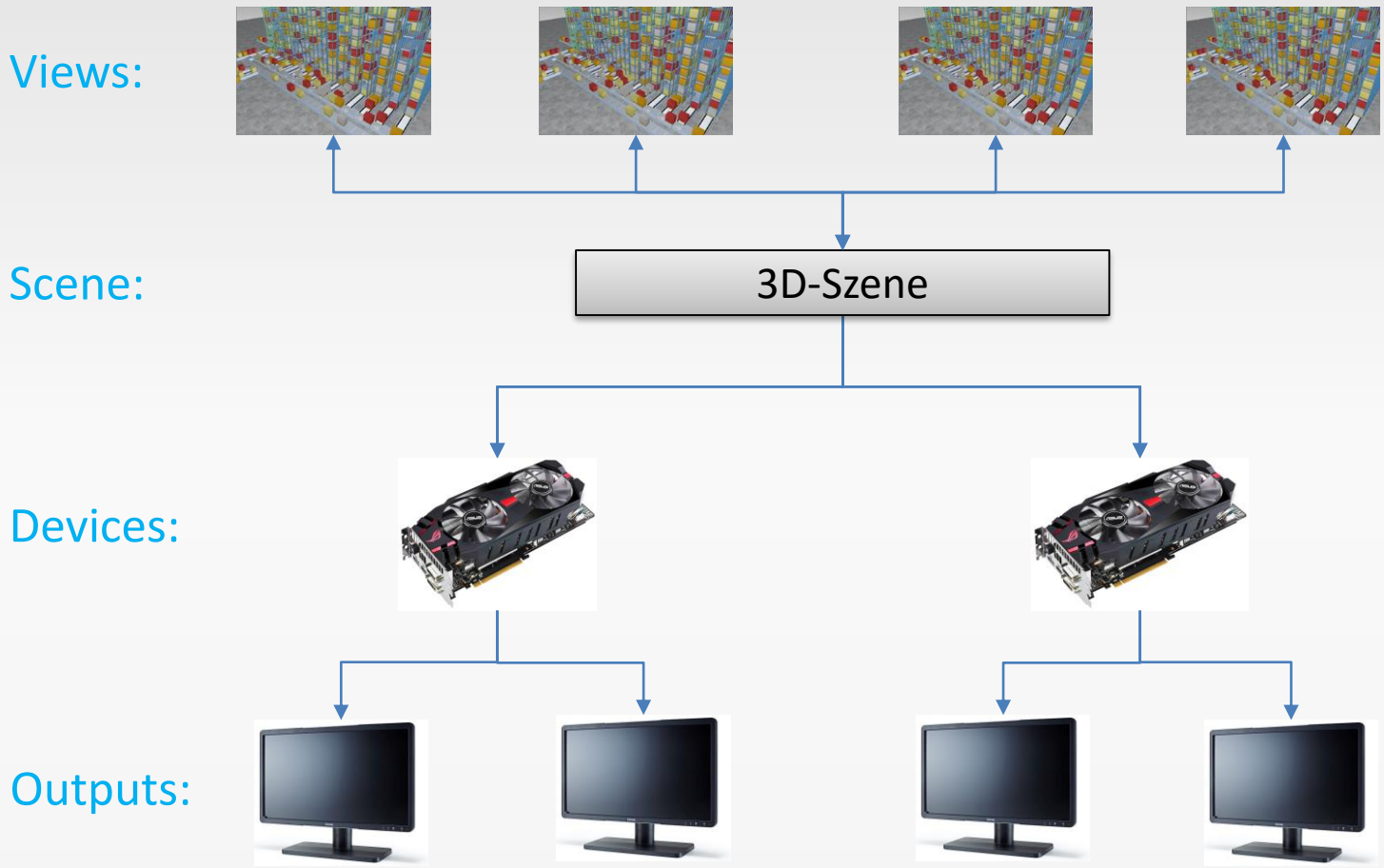
Herausforderung: Performance

- Zeit pro Frame (Bild): ca. **25ms** **(→ bei ~40 FPS)**
  ca. **11ms** **(→ bei VR, ~90 FPS)**

- Aufgaben pro Frame
  - Rendering
  - Div. Optimierungen (z. B. Render-Reihenfolge)
  - Hit-Testing (Mouse-Over)
  - Culling (Sichtbarkeits-Prüfungen)
  - Animationen
  - Kollissionsprüfung
  - Ressourcen laden / entladen
  - Gui-Synchronisierung
  - Uvm…

Übersicht Seeing#

Herausforderung: Hardware-Anbindung



3D-Programmierung in C#

Agenda

- Übersicht Seeing#
- Diverse Aspekte der Entwicklung
  - **Multiplatform**
  - Parallelisierung
  - Qualitätssicherung
  - 3D-Rendering
- Einstieg

Multiplatform

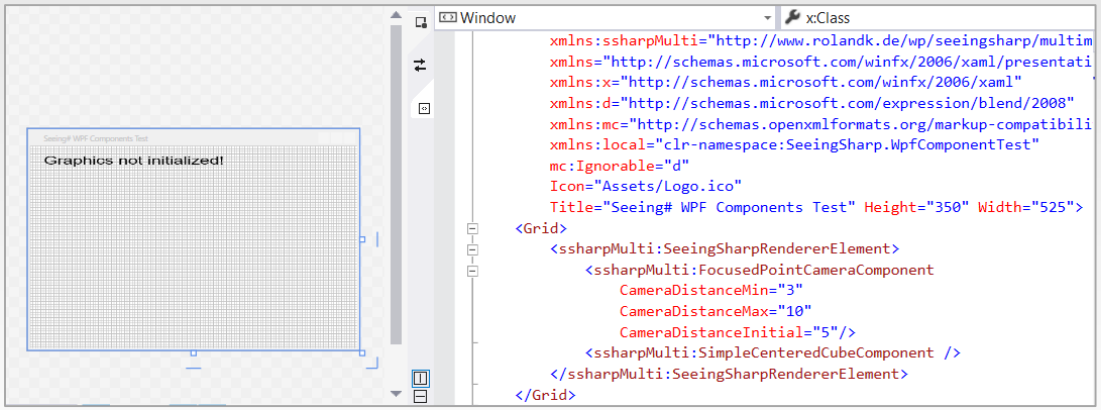
Mögliche Plattformen

- Windows.Forms
  - Desktop-App
  - ActiveX
- WPF
- Universal Apps / Windows Runtime
- In-Memory
  - Video-Rendering
  - Web
- Virtual Reality (z. B. Oculus Rift)

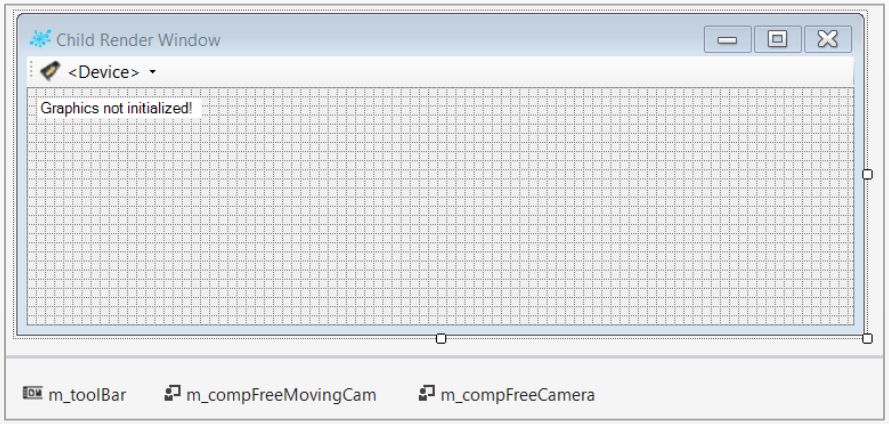
Multiplatform

Maßnahmen

- Ähnliche Standard-Controls für jedes UI-Framework
  - Xaml:



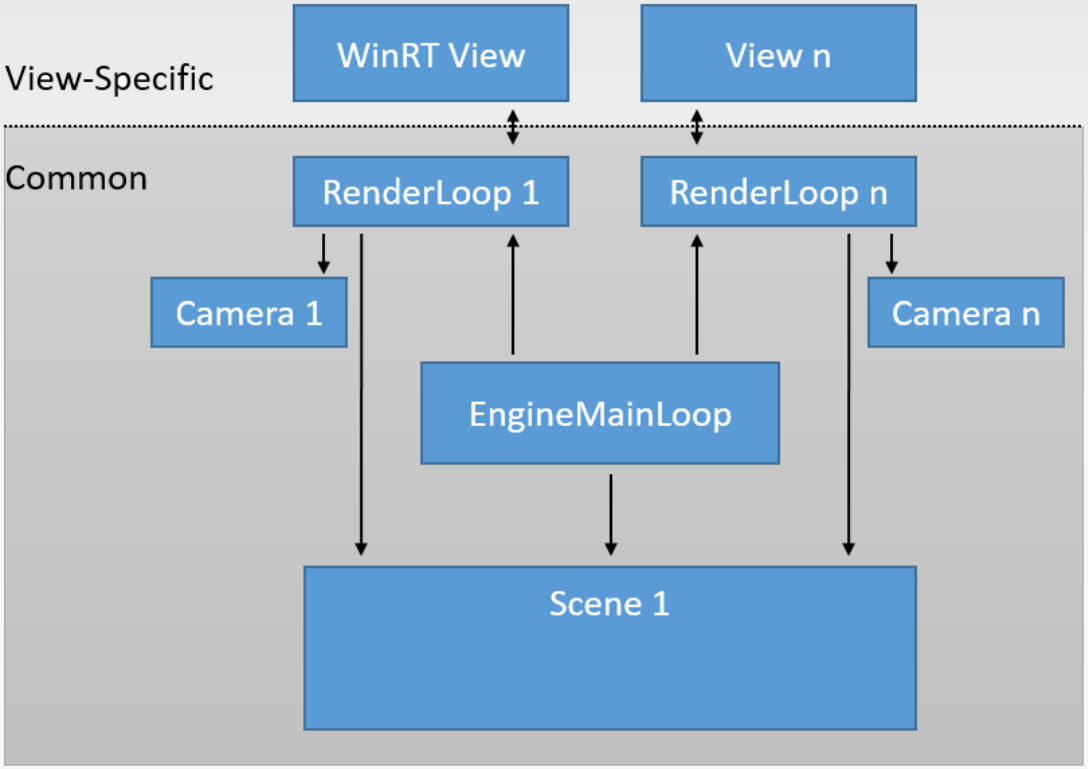
  - Forms:



Multiplatform

Maßnahmen

- Möglichst hoher Grad gemeinsamen Codes

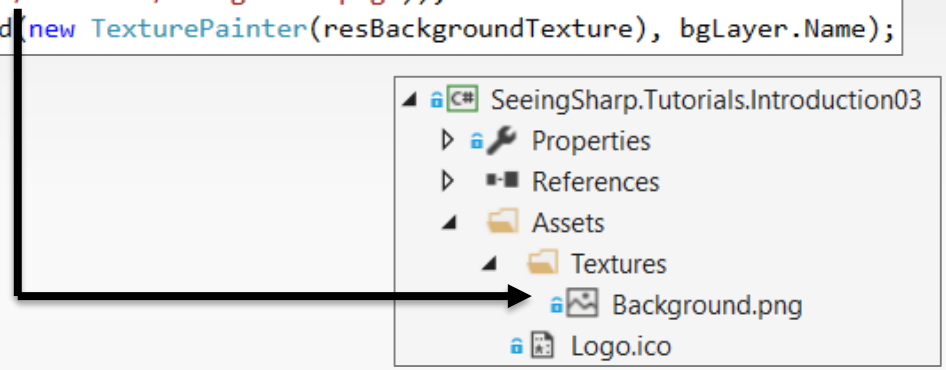


Multiplatform

Maßnahmen

- Abstraktion von Asset-Quellen

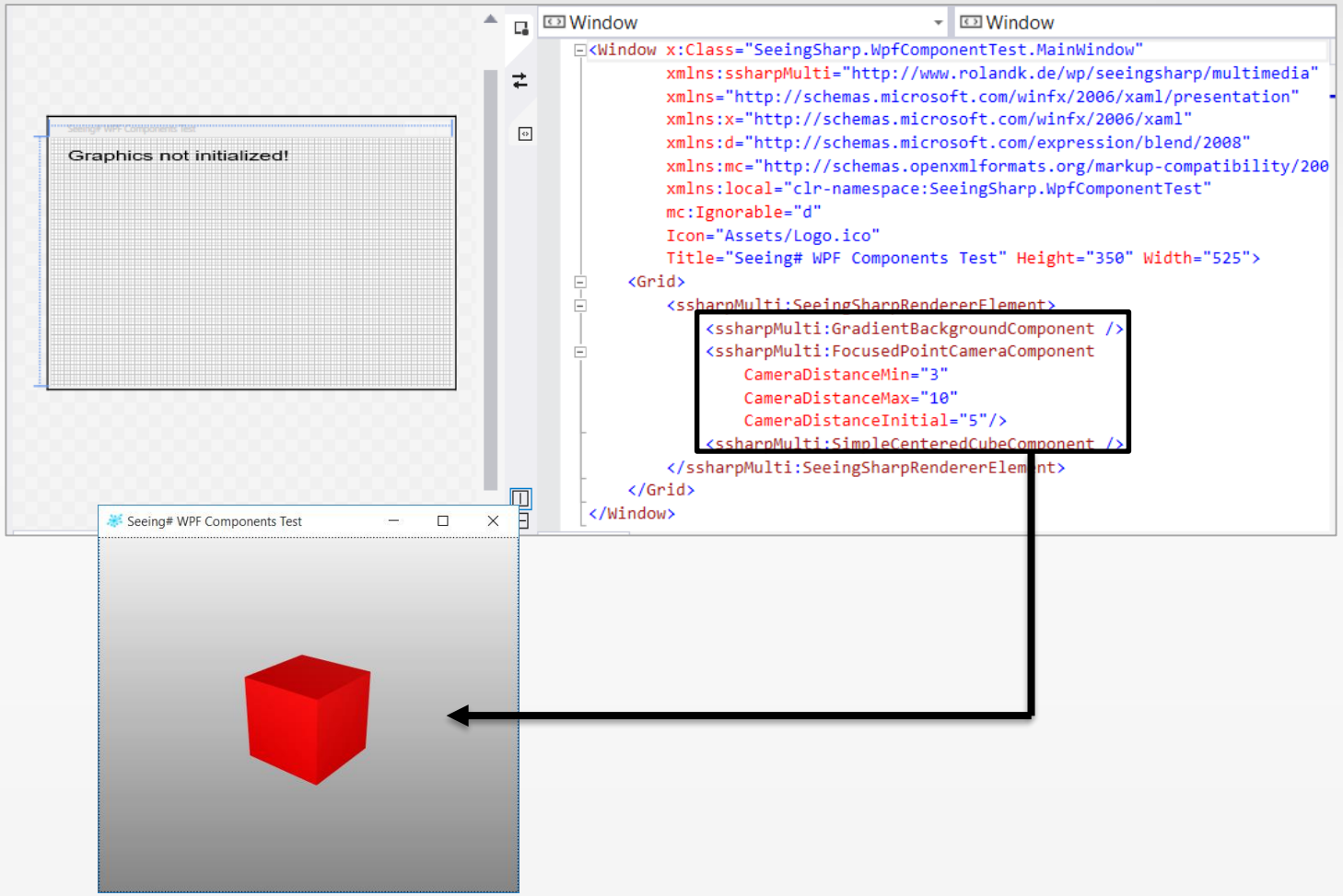
```
// Add the background texture painter to the BACKGROUND layer
var resBackgroundTexture = manipulator.AddTexture(
    new AssemblyResourceUriBuilder(
        "SeeingSharp.Tutorials.Introduction03",
        true,
        "Assets/Textures/Background.png"));
manipulator.Add(new TexturePainter(resBackgroundTexture), bgLayer.Name);
```



Multiplatform

Maßnahmen

- Gemeinsames Komponenten-System



3D-Programmierung in C#

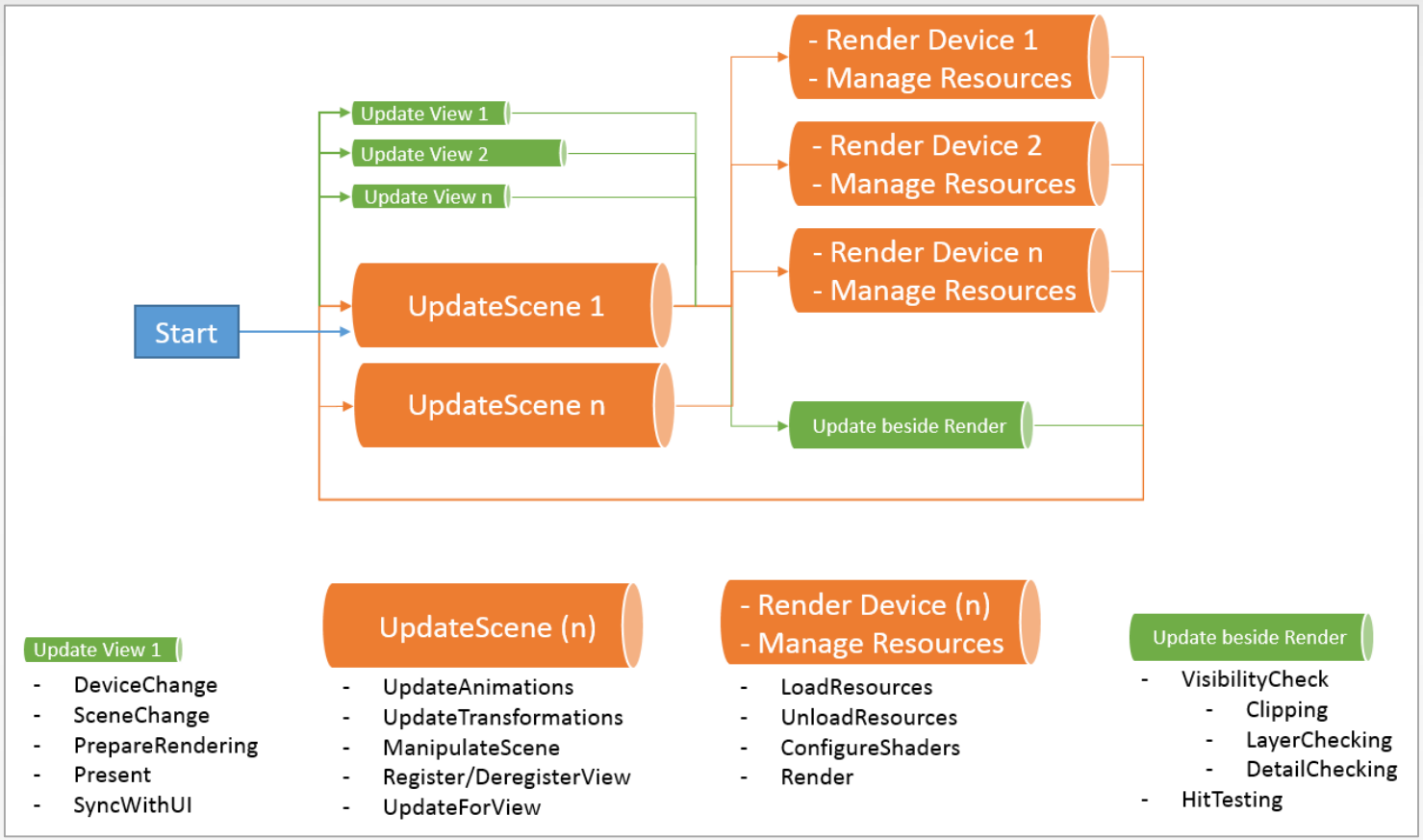
Agenda

- Übersicht Seeing#
- Diverse Aspekte der Entwicklung
  - Multiplatform
  - **Parallelisierung**
  - Qualitätssicherung
  - 3D-Rendering
- Einstieg

Parallelisierung

Maßnahmen

- 3D-Engine Hauptschleife per async-await
  (Jeder Zylinder ist ein Task)



Parallelisierung

Maßnahmen

- Synchronisierung mit Queues und async/await

```
private async void OnMainPage_Loaded(object sender, RoutedEventArgs e)
{
    if (m_panelPainter != null) { return; }

    // Attach the painter to the target render panel
    m_panelPainter = new SeeingSharpPanelPainter(this.RenderTargetPanel);
    m_panelPainter.RenderLoop.ClearColor = Color4.CornflowerBlue;

    // Build scene graph
    await m_panelPainter.Scene.ManipulateSceneAsync((manipulator) =>
    {
        // Create pallet geometry resource
        PalletType pType = new PalletType();
        pType.ContentColor = Color4.GreenColor;
        var resPalletGeometry
            () => new Geometry
```

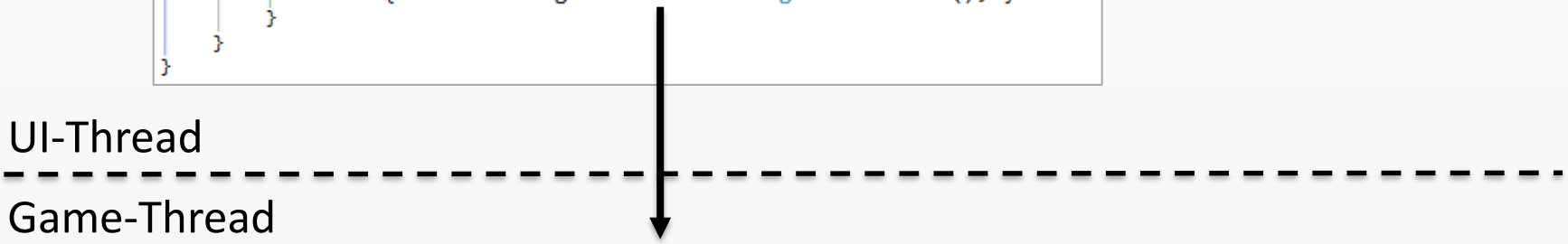
```
public Task PerformBeforeUpdateAsync(Action actionToInvoke)
{
    actionToInvoke.EnsureNotNull(nameof(actionToInvoke));

    TaskCompletionSource<bool> taskCompletionSource = new TaskCompletionSource<bool>();

    m_asyncInvokesBeforeUpdate.Enqueue(() =>
    {
        try
        {
            actionToInvoke();

            taskCompletionSource.TrySetResult(true);
        }
        catch (Exception ex)
        {
            taskCompletionSource.TrySetException(ex);
        }
    });

    return taskCompletionSource.Task;
}
```

Parallelisierung

Maßnahmen

- Messages nach Publish / Subscribe
  Besonderheit: Berücksichtigung Quell- und Ziel-Threads

```
public bool IsPaneVisible
{
    get { return m_isPaneOpened; }
    set
    {
        if(m_isPaneOpened != value)
        {
            m_isPaneOpened = value;
            RaisePropertyChanged();

            if (value) { base.Messenger.Publish<MessageMenuOpened>(); }
            else { base.Messenger.Publish<MessageMenuClosed>(); }
        }
    }
}
```




```
private void OnMessage_Received(MessageMenuClosed message)
{
    m_isMenuOpened = false;

    this.UpdateStates();
}
```

3D-Programmierung in C#

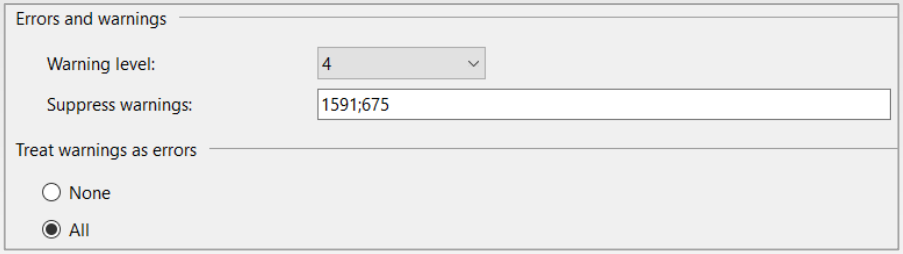
Agenda

- Übersicht Seeing#
- Diverse Aspekte der Entwicklung
  - Multiplatform
  - Parallelisierung
  - **Qualitätssicherung**
  - 3D-Rendering
- Einstieg

Qualitätssicherung

Maßnahmen

- Warnungen als Fehler behandeln



- Guard Clauses im Debug-Build

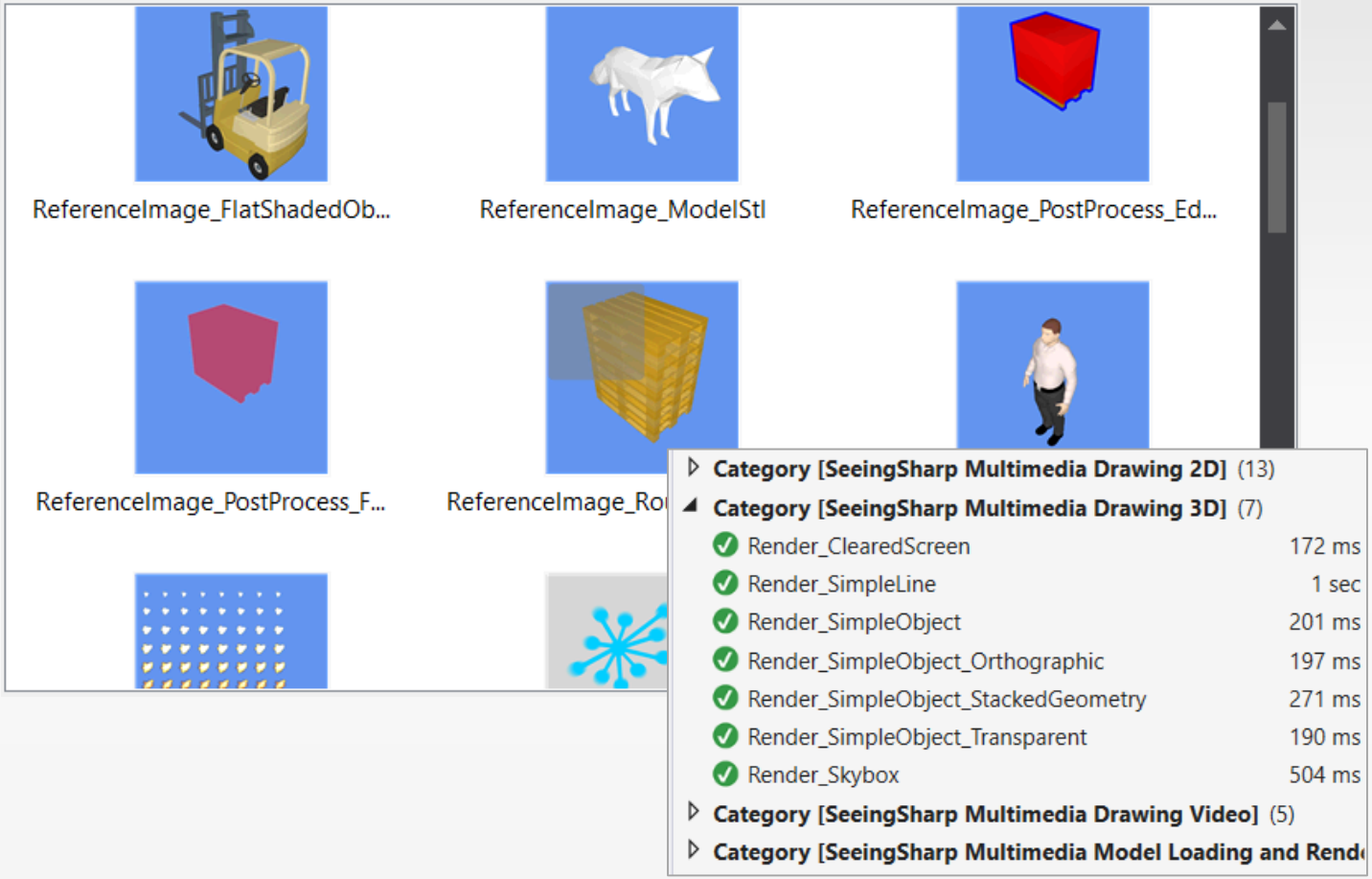
```
/// <summary>
/// Waits until the given object is visible on the given view.
/// </summary>
/// <param name="sceneObject">The scene object to be checked.</param>
/// <param name="viewInfo">The view on which to check.</param>
/// <param name="cancelToken">The cancellation token.</param>
public Task WaitUntilVisibleAsync(SceneObject sceneObject, ViewInformation viewInfo,
{
    sceneObject.EnsureNotNull(nameof(sceneObject));
    sceneObject.EnsureNotNull(nameof(viewInfo));
```

```
[Conditional("DEBUG")]
public static void EnsureNotNull(
    this object objParam, string checkedVariableName,
    [CallerMemberName]
    string callerMethod = "")
{
    if (string.IsNullOrEmpty(callerMethod)) { callerMethod = "Unknown"; }

    if (objParam == null)
    {
        throw new SeeingSharpCheckException(string.Format(
```

Qualitätssicherung

Maßnahmen

- Testautomatisierung: Rendern, dann Abgleich Referenz-Bild

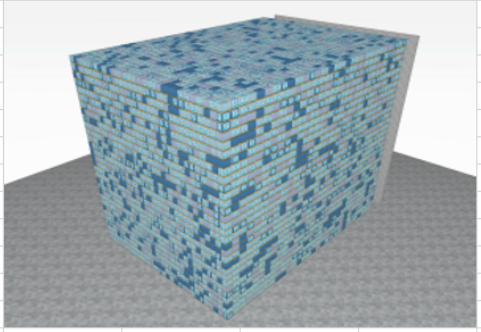


Qualitätssicherung

Maßnahmen

- Regelmäßiges Profiling → Ergebnisse dokumentieren!

Performance Paletten-Demo (27000 Paletten)		
Test 1		
Parameter	**Wert**	
Device-Level	Direct3D 11	
Shader-Model	5_0	
Antialiasing	Ja	
Anzahl Objekte	27003	
Grafikkarte	R9 200	
Prozessor	I7 3,5 Ghz	
Auflösung	1280x1024	
Name	**Duration**	
Graphics.Global.OneFrame	73,062	
Graphics.Global.Render (Device: AMD Radeon R9 200 Series)	59,590	
Graphics.Global.RenderAndUpdateBeside	59,634	
Graphics.Global.Update (Scene: 0)	13,257	- Jede Palette besteht aus 2 VertexStructures
Graphics.Global.UpdateAndPrepare	13,390	=> Erste ist die Palette selbst
Graphics.Global.UpdateBeside (Scene: 0)	6,169	=> Zweite ist der Inhalt (Blaue Quader, z. T. texturiert)
Graphics.Global.WaitTime	0,000	- Bei texturierten entstehen 2 VertexStructures pro HU
Graphics.RenderLoop.Present (Scene: 0, View: 1)	0,087	- Dadurch entstehen 2 DrawCalls für diese HUs
Graphics.RenderLoop.Render (Scene: 0, View: 1)	59,5904	



3D-Programmierung in C#

Agenda

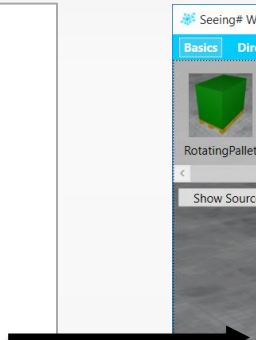
- Übersicht Seeing#
- Diverse Aspekte der Entwicklung
  - Multiplatform
  - Parallelisierung
  - Qualitätssicherung
  - **3D-Rendering**
- Einstieg

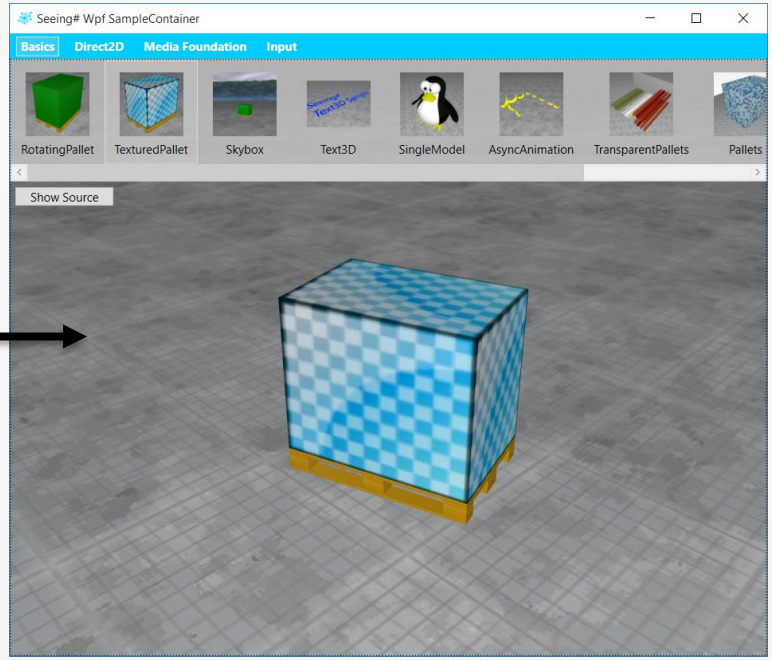
3D-Rendering

Basics

- Wichtigste Bestandteile beim 3D-Rendering
  - Material
  - Geometrie
  - Licht
  - Shader

```
public override async Task OnStartupAsync(RenderLoop targetRenderLoop)
{
    targetRenderLoop.EnsureNotNull(nameof(targetRenderLoop));

    // Build dummy scene
    Scene scene = targetRenderLoop.Scene;
    Camera3DBase camera = targetRenderLoop.Camera as Camera3DBase;

    await targetRenderLoop.Scene.ManipulateSceneAsync((manipulator) =>
    {
        // Create floor
        SampleSceneBuilder.BuildStandardFloor(
            manipulator, Scene.DEFAULT_LAYER_NAME);

        // Define texture and material resource
        var resTexture = manipulator.AddTexture(
            new AssemblyResourceUriBuilder(
                "SeeingSharp.Samples.Base", false,
                "Assets/Textures/SimpleTexture.png"));
        var resMaterial = manipulator.AddSimpleColoredMaterial(resTexture);

        // Create pallet geometry resource
        PalletType pType = new PalletType();
        pType.ContentMaterial = resMaterial;
        var resPalletGeometry = manipulator.AddResource<GeometryResource>(
            () => new GeometryResource(pType));

        // Create pallet object
        GenericObject palletObject = manipulator.AddGeneric(resPalletGeometry);
        palletObject.Color = Color4.GreenColor;
        palletObject.EnableShaderGeneratedBorder();
        palletObject.BuildAnimationSequence()
            .RotateEulerAnglesTo(new Vector3(0f, EngineMath.RAD_180DEG, 0f), TimeSpan.FromSeconds(2.0))
            .WaitFinished()
            .RotateEulerAnglesTo(new Vector3(0f, EngineMath.RAD_360DEG, 0f), TimeSpan.FromSeconds(2.0))
            .WaitFinished()
            .CallAction(() => palletObject.RotationEuler = Vector3.Zero)
            .ApplyAndRewind();
    });

    // Configure camera
    camera.Position = new Vector3(2f, 2f, 2f);
    camera.Target = new Vector3(0f, 0.5f, 0f);
    camera.UpdateCamera();
}
```



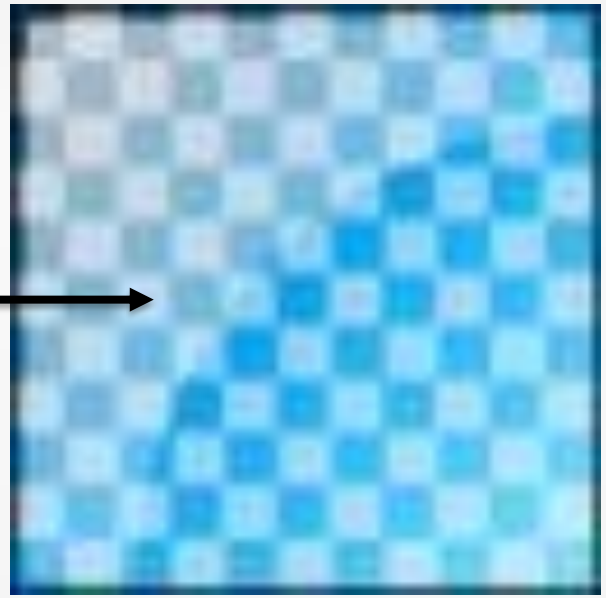


3D-Rendering

Material

- Definiert Texturen, Farben, Reflektionen, etc.

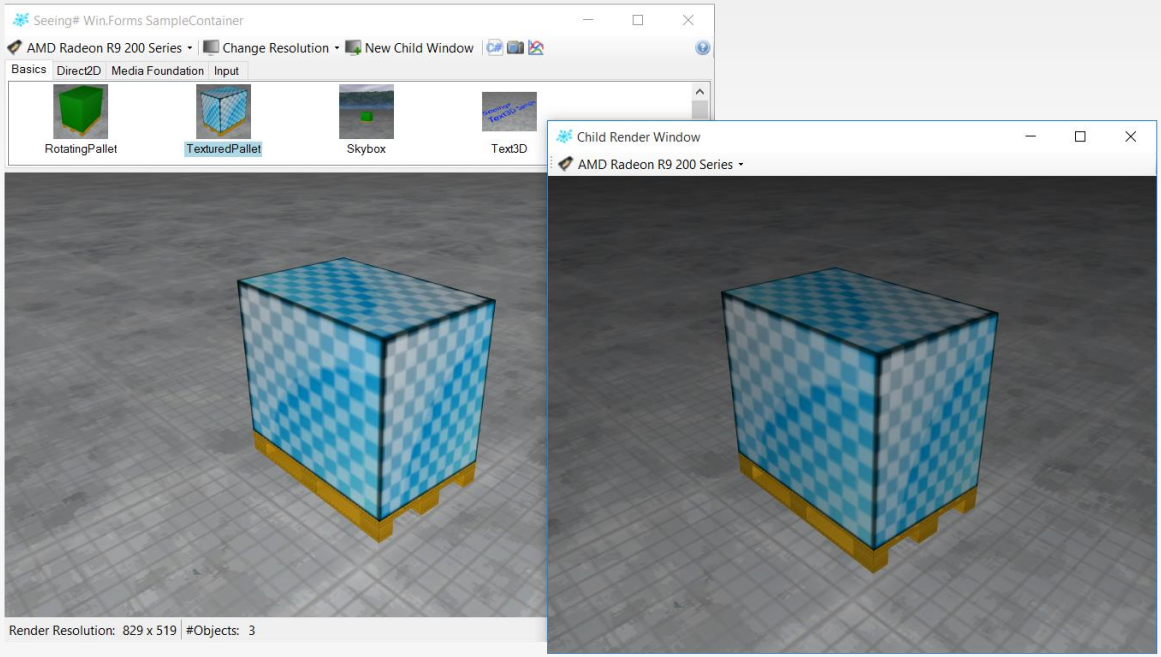
```
// Define texture and material resource
var resTexture = manipulator.AddTexture(
    new AssemblyResourceUriBuilder(
        "SeeingSharp.Samples.Base", false,
        "Assets/Textures/SimpleTexture.png"));
var resMaterial = manipulator.AddSimpleColoredMaterial(resTexture);
```



3D-Rendering

Licht

- In Seeing# aktuell fix
  - Kamera = Lichtquelle
  - Allgemeine Einstellungen je View



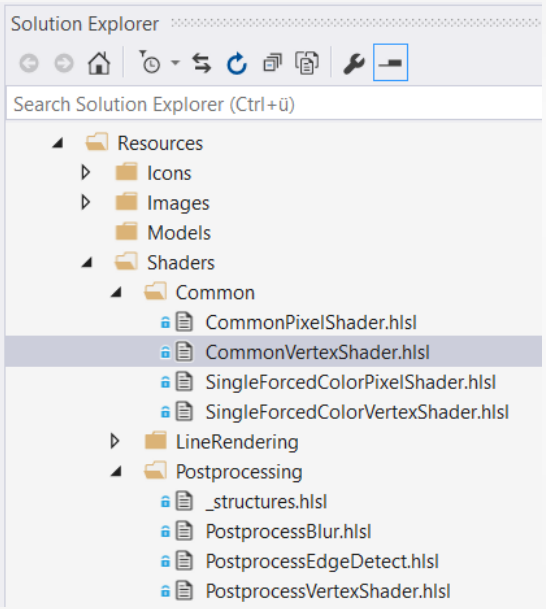
3D-Rendering

Shader

- Material bestimmt die aktiven Shader
- Für Rendering mind. benötigt:
  - Vertex Shader
  - Pixel Shader
- Shader direkt in Visual Studio eingebunden

CommonVertexShader.hlsl

```
//------------------------------------------------------------------------------
//Include all common items
#include "../_mainInclude.hlsl"

//------------------------------------------------------------------------------
//VertexShader implementation
PSInputStandard main(VSInputStandard input)
{
    PSInputStandard output = (PSInputStandard)0;

    // Calculate standard values (position and normal)
    float4x4 localWorldViewProj = mul(World, ViewProj);
    output.pos = mul(float4(input.pos.xyz, 1.0), localWorldViewProj);
    output.normal = normalize(mul(input.normal, (float3x3)World));

    // Interpolate texture coordinate based on xy-scaling
    if(AdjustTextureCoordinatesFactor > 0.0)
    {
        output.tex.xy = abs(input.tex.xy * (ObjectScaling.xy * input.normal.z + Objec
    }
    else
    {
        output.tex.xy = input.tex.xy;
    }

    // Apply object color for all color values with low alpha
```



3D-Rendering

Sonstiges

- Um was muss man sich noch Gedanken machen?
  - Koordinaten-System
  - Maßeinheiten
  - Dateiformate
  - Bedienung

3D-Programmierung in C#

Agenda

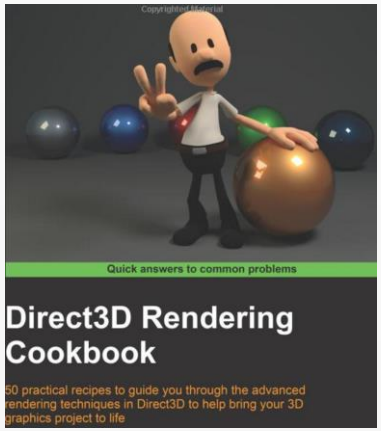
- Übersicht Seeing#
- Diverse Aspekte der Entwicklung
  - Multiplatform
  - Parallelisierung
  - Qualitätssicherung
  - 3D-Rendering
- **Einstieg**

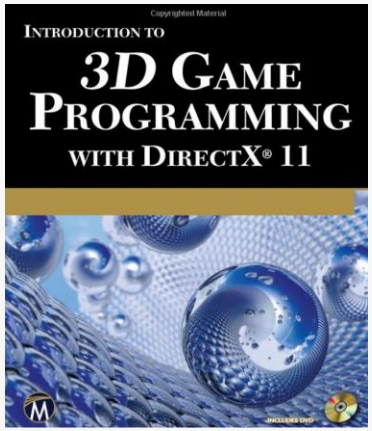
Einstieg

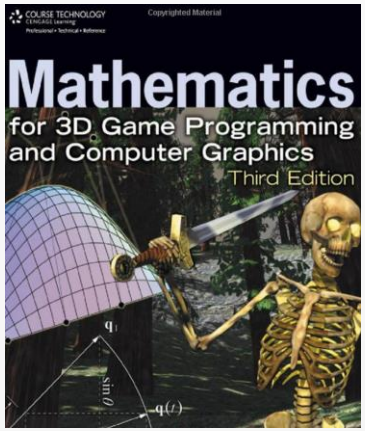
- Seeing# auf Github
  https://github.com/RolandKoenig/SeeingSharp
  https://github.com/RolandKoenig/SeeingSharp.Tutorials

- SharpDX: http://sharpdx.org

- Literatur:







Vielen Dank für eure Aufmerksamkeit!