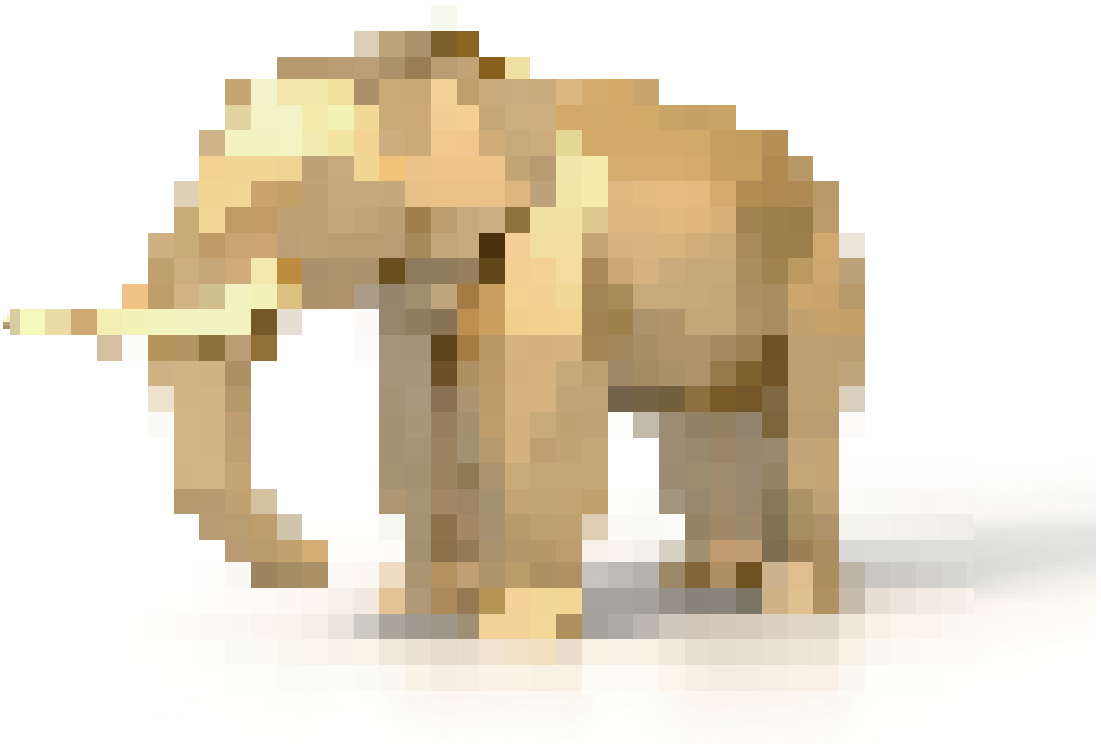
Tool Documentatie

Naam: Jin Zhou Liu
Github: 3DToSprite

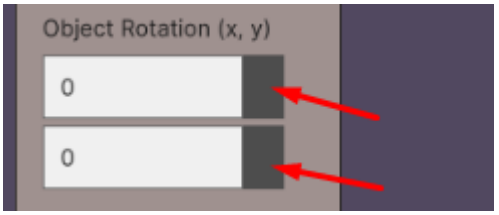


Concept

De tool is in principe een pixelator met wat extra functionaliteit. Geïnspireerd door Dead Cells, Het doel van de tool is om sneller 2D sprite animaties te kunnen maken en er makkelijker op te kunnen itereren. Dit doel wordt behaald doordat je een 3D model gebruikt en animaties voor het 3D model maakt. Vervolgens kun je dit in de tool converteren naar een 2D Sprite sheet, inclusief met de normal map data van het 3D model waardoor je ook gelijk lighting data hebt voor je sprites.

In het huidig stadium is de tool in bezit van de kernfuncties. Dit houdt in het kunnen pixeleren van een 3D model met een gekozen animatie. Deze wordt geëxporteerd in 2 .PNG's 1 Diffuse map en een Normal map voor de lighting data.

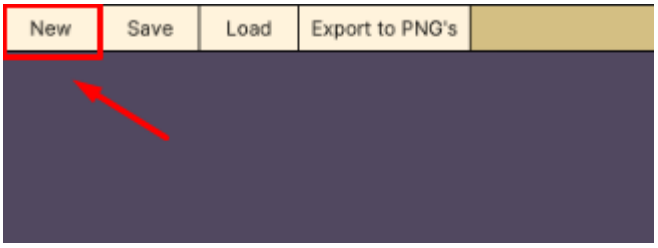
Verder is het mogelijk om de transform van het geïmporteerde model te veranderen. Dit kun je doen door het invoeren van een waarde in een veld.



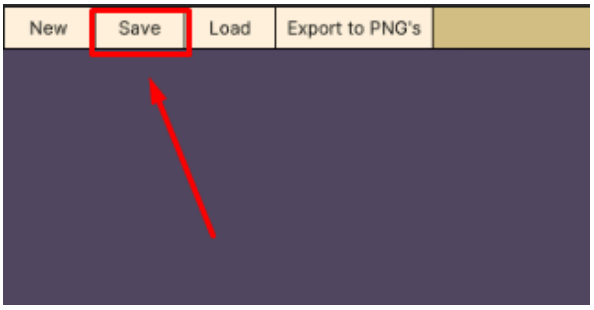
Ook Is het mogelijk bij velden met een zwarte rechthoek ernaast uw muis te gebruiken.
Als je erop klikt op het grijze gebied en het inhoud en daarna uw muis beweegt zal de waarde op die manier ook veranderen.

Voor de rotatie waardes zijn de velden gesloten tussen de 0 - 360. Dit betekent dat wanneer de waarde boven de 360 zou gaan hij weer terug naar 0 gaat. Dit gaat met behulp van een modulo operatie.

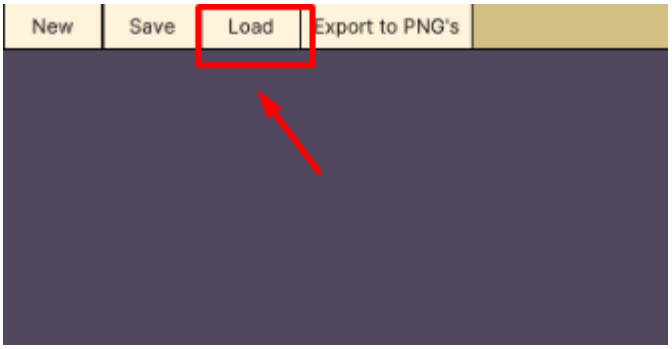
De bovenste taakbalk bevat een aantal knoppen. Met verschillende functionaliteiten.



Met de “New” knop kun je wat je huidig in je project hebt wegvegen. Dit functioneert als een soort reset knop. Dit vraagt niet of je je huidige data wilt opslaan.



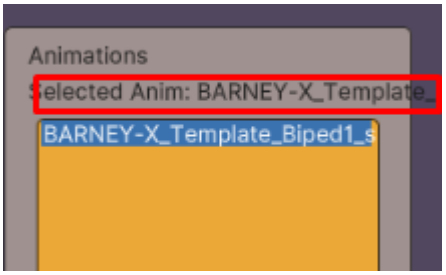
Je kunt alle input fields opslaan in een .json via de “Save” knop. Dit zal een file browser openen waarna je de path van het bestand kan kiezen. De data wordt opgeslagen in de ProjectData klasse en geserialiseerd naar .JSON, deze kan later ook weer worden uitgelezen in de ProjectData klasse.



Met de “Load” knop kun je een .JSON bestand kiezen die is gebaseerd op de ProjectData klasse. Deze laad alle relevante velden in. De transform data, hoe gepixeleert het resultaat zal zijn en de aantal frames.

Met de “Export to PNG’s” zal hij met de geselecteerde animatie een sprite sheet maken. Het resultaat is 1 Diffuse map en 1 Normal map van de animatie.

De ListView bevat alle animaties die ook in het .glb bestand zaten. Als je op een element klikt speelt de animatie automatisch af op het model.

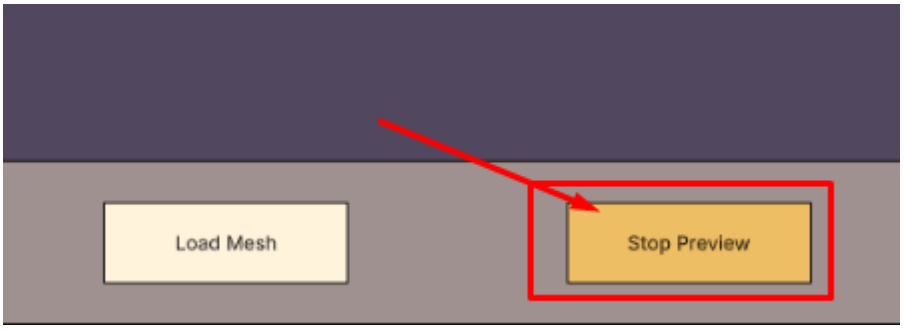


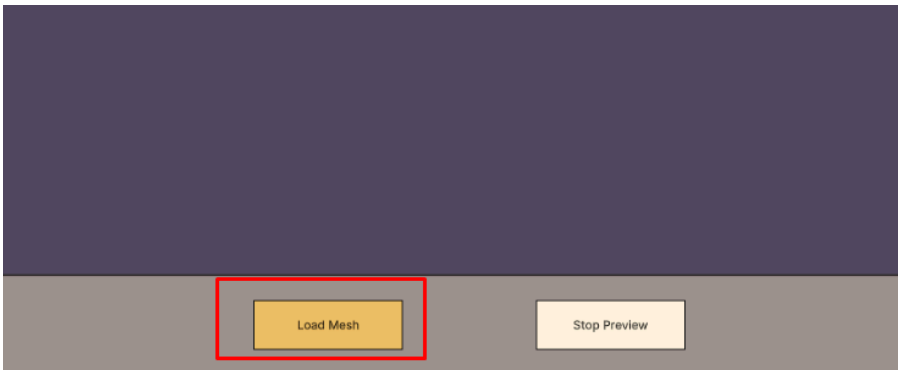
Hij laat ook zien wat de huidige geselecteerde animatie is. Deze zal worden gebruikt als je gaat exporteren.

Op dit moment wanneer je uit de ListView gaat stopt hij met animaties spelen totdat je de animatie weer uit de lijst klikt.

Als je wilt dat de animatie door blijft spelen moet je de “Preview Animation” klikken.

Dit functioneert als een toggle. Wanneer je erop klikt zal hij van state veranderen. Je kunt hem ook weer uit zetten door er nog een keer op te klikken. De tekst zal veranderen tussen “Preview Animation” of “Stop Preview”.





Met de “Load Mesh” knop kun je een .GLB bestand openen waarbij hij dus een nieuwe 3D mesh laad en de bijbehorende animatie lijst. Het zal in eerste instantie een file browser openen waarbij je de path naar jouw .GLB bestand doorgeeft.

Voor de button highlighting heb ik gebruik gemaakt van **USS** (Unity Style Sheets) pseudo classes. Deze schakelt automatisch wanneer hij ziet dat een object een wordt bedekt door de muis of ermee wordt ge interacteert.

UIController

Voor de UI van de tool had ik besloten om Unity's nieuw UI Toolkit te gebruiken. De Tool UI bevat een aantal Integer & Float fields en een ListView. Deze worden aangedreven via het script **<UIController>.**

```
VisualElement root = GetComponent<UIDocument>().rootVisualElement;

//Top Bar
Button newButton = root.Q<Button>(name: "ButtonNew"); // Clear out all fields, open file
Button saveButton = root.Q<Button>(name: "ButtonSave"); //Save URL to glb model + Transf
Button loadButton = root.Q<Button>(name: "ButtonLoad"); //Load JSON file
Button exportButton = root.Q<Button>(name: "ButtonExport"); //Export Normal Map.PNG or m

//Middle Bar
animationsListView = root.Q<ListView>(name: "AnimationsList");
selectedAnimation = root.Q<Label>(name: "SelectedAnimationTxt");

frameAmount = root.Q<IntegerField>(name: "FrameAmount");

cellSizeX = root.Q<IntegerField>(name: "CellSizeX");
cellSizeY = root.Q<IntegerField>(name: "CellSizeY");

rotationX = root.Q<FloatField>(name: "RotationX");
rotationY = root.Q<FloatField>(name: "RotationY");

scaleX = root.Q<FloatField>(name: "ScaleX");
scaleY = root.Q<FloatField>(name: "ScaleY");
scaleZ = root.Q<FloatField>(name: "ScaleZ");

//Drag Handlers
dragRotX = root.Q<VisualElement>(name: "DragRotX");
dragRotY = root.Q<VisualElement>(name: "DragRotY");

dragScaleX = root.Q<VisualElement>(name: "DragScaleX");
dragScaleY = root.Q<VisualElement>(name: "DragScaleY");
dragScaleZ = root.Q<VisualElement>(name: "DragScaleZ");

//Bottom Bar
Button buttonLoadMesh = root.Q<Button>(name: "ButtonLoadMesh");
buttonAnimationPreview = root.Q<Button>(name: "ButtonPreviewAnimation");

newButton.clicked += () => NewProject();
saveButton.clicked += () => SaveProjectData();
loadButton.clicked += () => LoadProjectData();
exportButton.clicked += () => Pixelate();

buttonLoadMesh.clicked += () => LoadMesh();
buttonAnimationPreview.clicked += () => PreviewAnimation();

RegisterFieldCallbacks();
InitiateAnimationsList();
```

Om de functionaliteit van de UIElementen te kunnen gebruiken moest er eerst een referentie worden gemaakt door ze te queryen in het bijbehorende UIDocument. Ik heb de hele UI in een UXML bestand gemaakt via de UI Builder van Unity dus kan ik alles uit hetzelfde document halen.

Buttons

```
//Bottom Bar
Button buttonLoadMesh = root.Q<Button>( name: "ButtonLoadMesh");
buttonAnimationPreview = root.Q<Button>( name: "ButtonPreviewAnimation");

newButton.clicked += () => NewProject();
saveButton.clicked += () => SaveProjectData();
loadButton.clicked += () => LoadProjectData();
exportButton.clicked += () => Pixelate();

buttonLoadMesh.clicked += () => LoadMesh();
buttonAnimationPreview.clicked += () => PreviewAnimation();
```

Alle interactieve delen van de UI worden gedreven via Events.

```
private void RegisterFieldCallbacks()
{
    //Cell Size
    cellSizeX.RegisterCallback<ChangeEvent<int>>(evt :ChangeEvent<int> =>
    {
        cellSize.x = evt.newValue;
    });

    cellSizeY.RegisterCallback<ChangeEvent<int>>(evt :ChangeEvent<int> =>
    {
        cellSize.y = evt.newValue;
    });

    //Rotation
    rotationX.RegisterCallback<ChangeEvent<float>>(evt :ChangeEvent<float> =>
    {
        rotation.x = evt.newValue;
        ApplyRotation();
    });
```

Voor de Input Fields moest ik een callback registeren. Omdat het ik alleen de nieuwe waarde wilde en de Input Fields zelf al de input sanitized heb ik anonymous functions gebruikt als callback.

Drag Input

```
dragRotX.RegisterCallback<MouseDownEvent>(CaptureMouse);
dragRotX.RegisterCallback<MouseUpEvent>(ReleaseMouse);
dragRotX.RegisterCallback<MouseMoveEvent>(evt:MouseMoveEvent =>
{
    if(!isDragging) return;
    float value = rotation.x;
    value += evt.mouseDelta.x * 0.75f;
    rotation.x = Mod(value, modulo:360);
    rotationX.value = rotation.x;
    ApplyRotation();
});
```

Voor de functionaliteit waarbij je de waarde in een veld kan veranderen door het slepen van je muis heb ik 3 callbacks per veld moeten registreren op een VisualElement.
Bij muisklik en het loslaten van de muisknop heb ik de volgende functies aan de events gekoppeld.

```
private void CaptureMouse(MouseDownEvent evt)
{
    Debug.Log(message:"Mouse down on: " + evt.target);
    evt.target.CaptureMouse();
    isDragging = true;
}
5 usages  SpeedCheese
private void ReleaseMouse(MouseUpEvent evt)
{
    Debug.Log(message:"Mouse down up: " + evt.target);
    evt.target.ReleaseMouse();
    isDragging = false;
}
```

Met “CaptureMouse()” zorg ik ervoor dat de VisualElement alle input van de muis blijft doorkrijgen. Deze krijgt hij normaliter alleen door als de muis bovenop het object zit. Hierdoor kan ik ook de muis weg van het object slepen en zal hij de input alsnog door krijgen.
Wanneer ik de muisknop loslaat geeft hij deze ook weer op en zal hij stoppen met de muis input lezen als hij niet bovenop het object staat.

ListView

```
private void InitiateAnimationsList()
{
    Func<VisualElement> makeListItem = () => new Label();
    Action<VisualElement, int> bindItem = (e, a) => (e as Label).text = animationClips[a].name;

    const int itemHeight = 16;

    animationsListView.itemsSource = animationClips;
    animationsListView.fixedItemHeight = itemHeight;
    animationsListView.makeItem = makeListItem;
    animationsListView.bindItem = bindItem;

    animationsListView.selectionType = SelectionType.Single;
    animationsListView.itemsChosen += SampleListAnimation;
}
```

De ListView neemt 2 delegates voor creatie van een nieuw item en om de item te verbinden met data uit de sourceList.

```
private void UpdateAnimationsList(AnimationClip[] animations)
{
    animationClips = animations.ToList();

    Action<VisualElement, int> bindItem = (e, a) => (e as Label).text = animationClips[a].name;

    animationsListView.itemsSource = animationClips;
    animationsListView.bindItem = bindItem;
    animationsListView.RefreshItems();
}
```

Als er een nieuw *.glb* bestand wordt geïmporteerd wordt de ListView geüpdatet en gevuld met nieuwe animaties.

IOHandler

De <IOHandler> class is een statische class. Dit maakt het wat makkelijker om het aan te roepen wanneer je het nodig hebt zonder een reference er naar toe te hebben. Ik zorg er wel voor dat er niks wordt opgeslagen in de statische class en vooral operaties uitvoer die waardes returnen. Beide <UIController> & <AnimationCapturer> maken gebruik van functies uit deze class. Verder is de class niet afhankelijk van mijn zelf geschreven scripts.

File I/O

Voor de GUI van het vinden van een bestand/directory wordt er gebruik gemaakt van de Standalone File Browser.

JSON

Voor het opslaan en laden van data heb ik een class geschreven die een aantal variabelen opslaat. Het serializeren zelf gaat via de JsonUtility library van Unity.

Runtime Imports

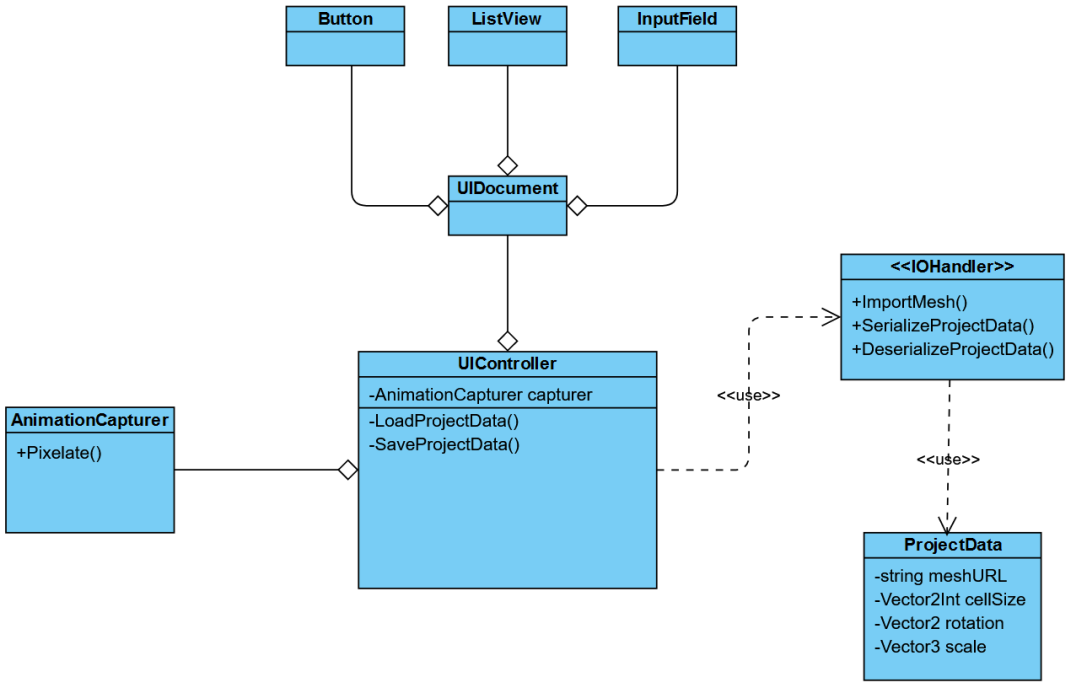
Importeren van *.glb* bestanden gaat via de gtlFast package voor Unity.

AnimationCapturer
De <AnimationCapturer> regelt het omzetten van het 3D model naar een sprite sheet.

De animation capturer berekent aan de hand van een gegeven FPS variabel hoeveel frames er nodig zijn in totaal. Vervolgens wordt de animatie gesampled via een frameTime en neemt de camera elke iteratie twee screenshots. 1 voor de Diffuse en 1 voor de Normals dit gebeurt door middel van een replacement shader.

Het uiteindelijke resultaat is 2 *.png* bestanden.

Class Diagram



Activity Diagram