Skinned 3D animaties met XNA Framework

Beweging nabootsen op een natuurlijke manier

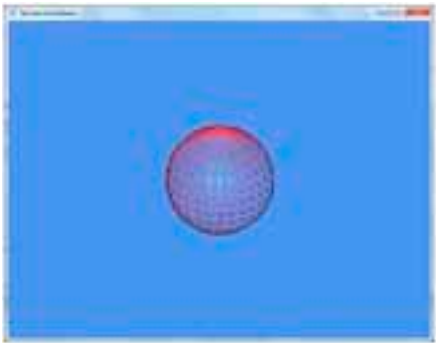
Kevin Lubbersen en Dennis Mijer

Het XNA Framework is in 2006 door Microsoft uitgebracht om game-ontwikkeling voor Windows, XBOX360 en de Zune toegankelijk te maken. In dit artikel gaan we 3D-animaties in XNA beschrijven. Een animatie is het bewegen van objecten. Aan de hand van een voorbeeld laten we zien hoe dit te realiseren is. Je moet wel de XNA Game Studio 3.1 geïnstalleerd te hebben om het voorbeeld te kunnen volgen.

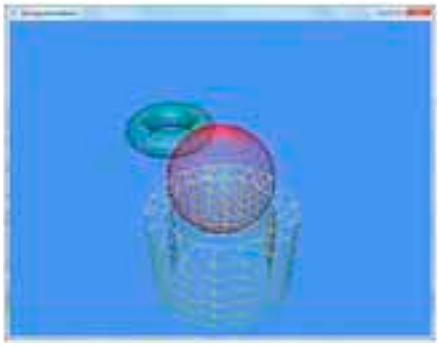
Voor de 3D-modellen

is een trial van Autodesk 3ds Max te downloaden. Vergelijkbare software is ook mogelijk, mits deze een export functionaliteit bevat om te exporteren naar .fbx.
Laten we beginnen met de theoretische basis van skinned animaties. Een model in XNA is een representatie van een scene die in een content creation tool (zoals Blender, Maya of 3D Studio Max) is gemaakt. Een model bestaat uit één of meerdere meshes. Zie Figuur 1 voor een voorbeeld van een model in XNA. Een mesh is het daadwerkelijke 3D object. Elke mesh binnen een model kan individueel worden getransformeerd, hiermee wordt bedoeld dat een mesh kan worden verplaatst, geschaald of gedraaid. Een vertex is een punt in een 3D wereld en heeft daarom een X, Y en Z coördinaat. Een mesh is opgebouwd uit vertices. Deze vertices zijn onderling met elkaar verbonden om zo een 3D model te vormen. Zie Figuur 2 voor een voorbeeld van een model met meerdere meshes in XNA.

Een transformatiematrix is een matrix die transformaties als verplaatsen, roteren en schalen, toepast op vertices. Een bone is een transformatiematrix. Een model kan één of meerdere bones bevatten. Een bone kan in een hiërarchie zijn geplaatst om zo een skelet te vormen. De bone is een essentieel onderdeel bij skinned animaties. Net als in het menselijk lichaam zijn het de bones in een model die bij animaties getransformeerd worden. Standaard heeft elke mesh één bone in XNA. Deze bone geeft de transformatie van deze mesh ten opzichte van de oorsprong van het hele model aan.
De skin is een essentieel onderdeel bij skinned animaties.
Een skin, of skin-modifier, koppelt één of meerdere bones aan een vertex uit een mesh. Op deze manier is te achterhalen welke bone invloed heeft op welk deel van de mesh. Een skin-deform shader kan vervolgens berekenen welke vertices uit een mesh moeten worden getransformeerd en hoe.
Een pivot is een draaipunt waar een mesh, of een deel van de mesh omheen draait. Bij skinned animaties, zoals beschreven in dit artikel, worden de bones gebruikt als draaipunten. Een shader, of Effect in XNA, bestaat in ieder geval uit een Vertexshader functie, een Pixelshader functie en één Technique die uit één Pass bestaat. Een basis Vertexshader functie transformeert de positie van elke vertex van object-space naar world-space. Een Pixelshader functie kan het model kleur geven of een materiaal, zoals hout, metaal, etc., nabootsen. Voor skinned animaties wordt een zogenaamde skin-deform shader gebruikt. Deze shader wordt verderop uitgelegd. Deze shader is geschreven in High Level Shading Language (HLSL).Het belangrijkste deel van de skindeform shader is de Vertexshader functie.
Net als de BasicEffect shader uit XNA heeft de skin-deform shader ook een aantal parameters die van buitenaf geset kunnen worden (zie Codevoorbeeld 1).



FIGUUR 1: MODEL.



FIGUUR 2: MODEL MET MEERDERE MESHES.

```
#define MaxBones 59

float4x4 World;
float4x4 View;
float4x4 Projection;
float4x4 Bones[MaxBones];

float4 Color;
float3 LightDirection1;
```

```
float3 LightDirection2;
float3 LightDirection3;
```

CODEVOORBEELD 1: SHADER PARAMETERS

De skin-deform shader neemt een World, View en Projection matrix als input. Daarnaast neemt het een array van matrices voor de bones, een kleur en drie licht richtingen als input. De array van matrices wordt gebruikt om de vertices die via de Skin aan een bone gekoppeld zijn te transformeren.

De Vertexshader functie van de skin-deform shader transformeert de binnenkomende vertex aan de hand van de transformatie matrices van de bones die invloed hebben op de betreffende vertex (zie Codevoorbeeld 2). De Vertexshader functie wordt voor elke vertex in een mesh uitgevoerd.

```
VS_OUTPUT VSMain(VS_INPUT input)
{
  VS_OUTPUT output;

  float4x4 skinTransform = 0;
  skinTransform += Bones[input.BoneIndices.x]
  * input.BoneWeights.x;
  skinTransform += Bones[input.BoneIndices.y]
  * input.BoneWeights.y;
  skinTransform += Bones[input.BoneIndices.z]
  * input.BoneWeights.z;
  skinTransform += Bones[input.BoneIndices.w]
  * input.BoneWeights.w;

  float4 position = mul(input.Position,
  skinTransform);
  position = mul(position, World);
  position = mul(position, View);
  position = mul(position, Projection);

  float3 normal = mul(input.Normal, skin-
  Transform);
  normal = mul(normal, World);
  normal = normalize(normal);

  output.Position = position;
  output.Normal = normal;

  return output;
}
```

CODEVOORBEELD 2: VERTEXSHADER FUNCTIE.

Als eerste wordt er een skin-transform matrix opgebouwd. Deze wordt opgebouwd aan de hand van de transformatiematrices van de verschillende bones in de mesh. In deze skindeform shader is de BoneIndices een float4, daardoor kan elke vertex door maximaal vier bones worden beïnvloed.

Nadat de skin-transform matrix is opgebouwd wordt de vertex getransformeerd met deze matrix. Vervolgens wordt de vertex getransformeerd van object-space naar world-space en worden de View en Projection matrices erop toegepast. De View matrix transformeert de vertices zodat er vanuit de camerapositie gekeken word. De Projection matrix transformeert de 3D wereld naar je 2D scherm.

De normaal van de vertex wordt ook getransformeerd met de skin-transform matrix en vervolgens ook van object-space naar world-space getransformeerd.

De input voor de Vertexshader functie kan worden gedefinieerd als struct (zie Codevoorbeeld 3). Dit is de definitie van één vertex uit een model.

```
struct VS_INPUT
{
  float4 Position : POSITION0;
  float4 BoneIndices : BLENDINDICES0;
  float4 BoneWeights : BLENDWEIGHT0;
  float3 Normal : NORMAL0;
};
```

CODEVOORBEELD 3: VERTEXSHADER INPUT.

De Vertexshader input struct bevat de positie van één vertex (Position), vier indices van de maximaal vier bones die invloed hebben op deze vertex (BoneIndices), de hoeveelheid invloed die een bepaalde bone op deze vertex heeft (BoneWeights), en de normaal van de vertex (Normal).

De output van de Vertexshader functie kan worden gedefinieerd als een struct (zie Codevoorbeeld 4). Deze struct is tevens de input voor de Pixelshader functie.

FIGUUR 3: KEYFRAME 1

FIGUUR 4: KEYFRAME 2.

```
struct VS_OUTPUT
{
  float4 Position : POSITION0;
  float3 Normal : TEXCOORD0;
};
```

CODEVOORBEELD 4: VERTEXSHADER OUTPUT

De Vertexshader output struct bevat de positie van de vertex (Position), dit is de positie die in de Vertexshader functie is getransformeerd van object space naar world space. Daarnaast bevat de struct de normaal van de vertex (Normal), dit is de normaal die in de Vertexshader functie is getransformeerd van object space naar world space.

Keyframe animatie

Een keyframe animatie is tijd gebonden en bestaat uit keyframes. Een keyframe bevat de pose van de bones op een bepaald tijdstip. In de tijd tussen twee keyframes worden de bones lineair getransformeerd om zo een animatie te vormen (Figuur 3 en 4).

Standaard staan alle vertices van een model in object space. Dat wil zeggen dat de positie van elke vertex relatief is aan de oorsprong van het model. Als je een game bouwt wil je de vertices (en daarmee het model) relatief aan de oorsprong van je spelwereld hebben, in world-space dus (Figuur 5 en 6).

Uitwerking

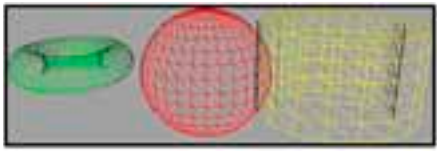
Nu gaan we stap voor stap in op hoe skinned animaties binnen XNA zijn te implementeren.

Pivots, oftewel draaipunten, zijn een belangrijk onderdeel van skinned animaties. Een pivot wordt geïmplementeerd met een translatiematrix.

Eerst moet er een model in een content creation tool gemaakt worden. Dit model kan op de normale manier worden ingeladen en getekend in XNA (Codevoorbeeld 5 en Figuur 7).

```
private Model _model;
private Matrix _world, _view, _projection;
private Matrix[] _transforms;

protected override void Initialize()
{
    _world = Matrix.Identity;
    _view = Matrix.CreateLookAt(new
    Vector3(200f, 200f, 200f),
            Vector3.Zero, Vector3.Up);
    _projection = Matrix.CreatePerspective-
    FieldOfView(
                    MathHelper.PiOver4,
                    GraphicsDevice.Viewport.
                    AspectRatio,
                    1f, 10000f);
    base.Initialize();
}

protected override void LoadContent()
{
    _model = Content.Load<Model>("Box");
    _transforms = new Matrix[_model.
    Bones.Count];
    _model.CopyAbsoluteBoneTransformsTo
    (_transforms);
}


protected override void Draw(GameTime
gameTime)
{
    GraphicsDevice.Clear(Color.Gray);

    foreach(ModelMesh mesh in _model.Meshes)
    {
```

FIGUUR 5: OBJECT SPACE.



FIGUUR 6: WORLD SPACE.

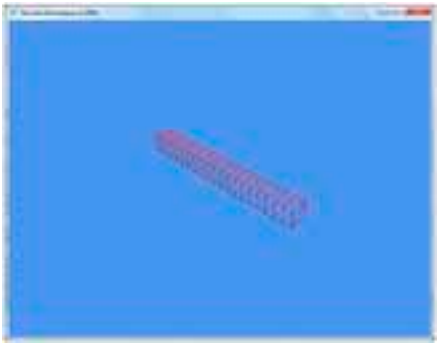
```
foreach(BasicEffect effect in mesh.Effects)
        {
            effect.EnableDefault-
            Lighting();
            effect.DiffuseColor =
            Color.Red.ToVector3();
            effect.World = _transforms
[mesh.ParentBone.Index] *
                            _world;
            effect.View = _view;
            effect.Projection = _
            projection;
        }
        mesh.Draw();
    }
    base.Draw(gameTime);
}
```

CODEVOORBEELD 5: MODEL LADEN IN XNA.

Pivots, oftewel draaipunten, zijn een belangrijk onderdeel van skinned animaties. Een pivot voor een model kan worden geïmplementeerd met een translatiematrix. Een model om een pivot draaien gaat via de volgende stappen:

- Het model wordt vanaf zijn huidige positie naar de oorsprong van de wereld getransformeerd.
- Het model wordt gedraaid.
- Het model wordt teruggeplaatst naar zijn oorspronkelijke positie.

```
private Matrix _pivot;
private Matrix _transform;
private Model _cube;

protected override void Initialize()
{
    …
    _pivot = Matrix.CreateTranslation(new
    Vector3(-132f, 50f, 0f));
    _transform = Matrix.Identity;
    base.Initialize();
}

protected override void LoadContent()
{
```



FIGUUR 7: MODEL IN XNA

Om gebruik te maken van skinned animaties moet de skin-deform shader in de LoadContent methode aan de mesh worden gekoppeld.

```
    _model = Content.Load<Model>("Box");
    _cube = Content.Load<Model>("Pivot");
}

protected override void Update(GameTime
gameTime)
{
    _transform *= Matrix.Invert(_pivot);
    _transform *= Matrix.CreateRotationZ
    (MathHelper.ToRadians(1f));
    _transform *= _pivot;

    base.Update(gameTime);
}

protected override void Draw(GameTime
gameTime)
{
    GraphicsDevice.Clear(Color.Cornflower-
    Blue);
    GraphicsDevice.RenderState.FillMode =
    FillMode.WireFrame;

    foreach(ModelMesh mesh in _model.
    Meshes)
    {
        foreach(BasicEffect effect in
        mesh.Effects)
        {
            effect.EnableDefaultLighting();
            effect.DiffuseColor = Color.
            Red.ToVector3();
           effect.World = mesh.ParentBone.
          Transform * _world * _transform;
            effect.View = _view;
           effect.Projection = _projection;
        }
        mesh.Draw();
    }

    GraphicsDevice.RenderState.FillMode =
    FillMode.Solid;
    foreach(ModelMesh mesh in _cube.
    Meshes)
    {
        foreach(BasicEffect effect in
        mesh.Effects)
        {
            effect.EnableDefault-
            Lighting();
            effect.DiffuseColor = Color.
            Black.ToVector3();
            effect.World = mesh.Parent-
            Bone.Transform * _world * _
            pivot;
            effect.View = _view;
            effect.Projection =
            _projection;
        }
        mesh.Draw();
    }

    base.Draw(gameTime);
}
```

CODEVOORBEELD 6: PIVOT.

Als je in een content creation tool een scene met meerdere meshes maakt en je zou de bone transformatie van elke mesh niet meenemen dan worden alle meshes in de oorsprong getekend (Figuur 9). Door de bone transformaties van de meshes wel mee te nemen, worden alle meshes op de juiste plaats gezet (Figuur 10).

De bones van de meshes kunnen ook manueel worden getransformeerd om zo de meshes los te verplaatsen.
In de meeste content creation tools kunnen ook handmatig bones worden toegevoegd aan een mesh. Deze bones kunnen in XNA ook worden getransformeerd, echter zonder skin-deform shader is dit visueel niet zichtbaar.
Bones kunnen ook in een hiërarchie worden geplaatst om zo een skelet te vormen. Als bijvoorbeeld de bovenarm van een model wordt gedraaid, moeten alle bones die onder deze bone in de hiërarchie vallen ook worden gedraaid.
Om gebruik te maken van skinned animaties moet de skin-deform shader aan de mesh gekoppeld worden. Dit kan worden gedaan in de LoadContent methode nadat het model is geladen (Codevoorbeeld 7).

```
Effect effect = Content.Load<Effect>
("SkinnedModel");
foreach(ModelMesh mesh in _model.Meshes)
{
    foreach(ModelMeshPart part in mesh.
    MeshParts)
    {
        part.Effect = effect;
    }
}
```

CODEVOORBEELD 7: SHADER KOPPELEN.

Nadat de shader aan het model is gekoppeld kunnen bones individueel worden getransformeerd. De transformaties worden door de shader ook op de mesh toegepast. Het ophalen van de bone-transformaties uit een model kan het best met een Model Processor gedaan worden.
Maak een nieuw Content Pipeline Extension Library project aan. Voeg aan dit nieuwe project een nieuwe class toe.
Laat de ContentProcessor class overerven van ModelProcessor (Microsoft.Xna.Framework.Content.Pipeline.Processors.ModelProcessor). Override de Process methode van de ModelProcessor (Codevoorbeeld 8).

```
[ContentProcessor]
public class SkinnedModelProcessor :
ModelProcessor
{
    public override ModelContent Process
    (NodeContent input,
ContentProcessor-Context context)
    {
        BoneContent skeleton = Mesh-
        Helper.FindSkeleton(input);
        IList<BoneContent> bones =
        MeshHelper.FlattenSkeleton
        (skeleton);

        Dictionary<string, Matrix> skin-
        Transforms =
new Dictionary
<string, Matrix>(bones.Count);
       Dictionary<string, Matrix> pivots =
new Dictionary<string, Matrix>
(bones.Count);
        Matrix rootTransform =
        FindRootTransform(input);

        Vector3 scale;
        Quaternion rotation;
        Vector3 translation;
        rootTransform.Decompose(out scale,
```

FIGUUR 8: PIVOT.

FIGUUR 9: MODEL ZONDER BONE TRANSFORMATIES.

FIGUUR 10: MODEL MET BONE TRANSFORMATIES.

```
        out rotation, out translation);
        rootTransform = Matrix.
        CreateFromQuaternion(rotation) *
                        Matrix.
                        CreateScale(scale);

        foreach(BoneContent bone in bones)
        {
            skinTransforms.Add
            (bone.Name, rootTransform);
            pivots.Add(bone.Name, bone.
            AbsoluteTransform);
        }

        ModelContent model =
        base.Process(input, context);
        model.Tag = new SkinnedModel-
        Data(skinTransforms, pivots);
        return model;
    }

    private Matrix FindRootTransform
    (NodeContent input)
    {
        Matrix output = Matrix.Identity;
        MeshContent mesh = input as
        MeshContent;

        if(mesh == null)
        {
            foreach(NodeContent child
            in input.Children)
            {
                mesh = child as Mesh-
                Content;
                if(mesh != null)
                {
                    output = mesh.
                    {
                        AbsoluteTransform;
                        break;
                    }
                    else
                    {
                        output = FindRoot-
```

```
                        Transform(child);
                    }
                }
            }
            else
            {
                output = mesh.Absolute-
                Transform;
            }

            return output;
        }
    }

    public class SkinnedModelData
    {
        [ContentSerializer]
        private Dictionary<string, Matrix> _
        skinTransforms;

        [ContentSerializer]
        private Dictionary<string, Matrix> _
        pivots;

        private SkinnedModelData()
        {
        }

        public SkinnedModelData(Dictionary
        <string, Matrix> skinTransforms,
                                Dictionary
                                <string,
                                Matrix>
                                pivots)
        {
            _skinTransforms = skinTransforms;
            _pivots = pivots;
        }

        public Matrix GetSkinTransform
        (string boneName)
        {
            return _skinTransforms[boneName];
        }

        public Matrix GetPivot(string boneName)
        {
            return _pivots[boneName];
        }

        public void SetSkinTransform(string
        boneName, Matrix skinTransform)
        {
            _skinTransforms[boneName] =
            skinTransform;
        }

        public void SetPivot(string boneName,
        Matrix pivot)
        {
            _pivots[boneName] = pivot;
        }

        public Matrix[] GetSkinTransforms()
        {
            Matrix[] result = new Matrix[_
            skinTransforms.Values.Count];
            _skinTransforms.Values.CopyTo
            (result, 0);
            return result;
        }

        public Matrix[] GetPivots()
        {
            Matrix[] result = new Matrix
            [_pivots.Values.Count];
            _pivots.Values.CopyTo(result, 0);
            return result;
        }
    }
```

CODEVOORBEELD 8: SKINNED MODEL PROCESSOR.

De processor zoekt eerst het skelet in het model. Vervolgens worden alle bones uit dit skelet gehaald. De basistransformatie van het hele model wordt gezocht om eventuele transformaties vanuit de content creation tool mee te nemen.
Als skin-transform wordt standaard de basistransformatie genomen. Op deze manier staat het model aan het begin in een neutrale positie. Als draaipunt voor elke bone wordt de transformatie van de bone genomen. De transformatie van de bone is namelijk de translatie ten opzichte van de oorsprong van het model en is daarmee uitermate bruikbaar als pivot. Alle data (de

De processor zoekt eerst het skelet in het model. Vervolgens worden alle bones uit dit skelet gehaald.

Al met al zijn skinned animaties goed te realiseren. Door de bone matrices goed te gebruiken zijn de pivots geen probleem.

skin-transformaties en de pivots) worden in de Tag property van het Model bewaard, zodat deze in de code van de game weer kunnen worden opgevraagd.
Voeg de zojuist gecreëerde Content Pipeline Extension Library als referentie toe aan het Content project van de game en aan de game zelf. Selecteer de nieuwe Model Processor als Content Processor voor het model.
De bones kunnen nu individueel getransformeerd worden door de Skin-transform met de naam van de bone uit de Tag van het model te halen en deze aan te passen (Codevoorbeeld 9).

```
private SkinnedModelData _modelData;

protected override void LoadContent()
{
    _spriteBatch = new SpriteBatch(GraphicsDevice);
    _model = Content.Load<Model>("Box");
    _modelData = (SkinnedModelData) _model.Tag;

    Effect effect = Content.Load<Effect>("SkinnedModel");
    foreach(ModelMesh mesh in _model.Meshes)
    {
        foreach(ModelMeshPart part in mesh.MeshParts)
        {
            part.Effect = effect;
        }
    }
}

protected override void Update(GameTime gameTime)
{
    TransformBone("Bone03", Matrix.CreateRotationZ(MathHelper.ToRadians(1f)));

    base.Update(gameTime);
}

protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    Matrix[] skinTransforms = _modelData.GetSkinTransforms();
    Matrix[] pivots = _modelData.GetPivots();

    foreach(ModelMesh mesh in _model.Meshes)
    {
        foreach(Effect effect in mesh.Effects)
        {
            effect.Parameters["World"].SetValue(_world);
            effect.Parameters["View"].SetValue(_view);
            effect.Parameters["Projection"].SetValue(_projection);
            effect.Parameters["Bones"].SetValue(skinTransforms);

            effect.Parameters["Color"].SetValue(Color.Red.ToVector4());
            effect.Parameters["LightDirection1"].SetValue(new Vector3(1f, 1f, 1f));
            effect.Parameters["LightDirection2"].SetValue(new Vector3(-1f, 1f, -1f));
        }
        mesh.Draw();
    }

    base.Draw(gameTime);
}

public void TransformBone(string boneName, Matrix transform)
{
    ModelBone bone = _model.Bones[boneName];
    Matrix skinTransform = _modelData.GetSkinTransform(boneName);
    Matrix bonePivot = _modelData.GetPivot(boneName);
    Matrix pivot = bonePivot;

    skinTransform *= Matrix.Invert(pivot);
    skinTransform *= transform;
    skinTransform *= pivot;

    bonePivot *= Matrix.Invert(pivot);
    bonePivot *= transform;
    bonePivot *= pivot;

    _modelData.SetSkinTransform(boneName, skinTransform);
    _modelData.SetPivot(boneName, pivot);
}
```

CODEVOORBEELD 9: BONE TRANSFORMEREN

Skinned Animaties met Hiërarchie

Om alle child bones in de hiërarchie ook te transformeren hoeft alleen de methode die de skin-transform toepast te worden aangepast (Codevoorbeeld 10).

```
public void TransformBone(string boneName, Matrix transform)
{
    TransformBone(boneName, _modelData.GetPivot(boneName), transform);
}

private void TransformBone(string boneName, Matrix pivot,
                           Matrix transform)
{
    ModelBone bone = _model.Bones[boneName];
    Matrix skinTransform = _modelData.GetSkinTransform(boneName);
    Matrix bonePivot = _modelData.GetPivot(boneName);

    skinTransform *= Matrix.Invert(pivot);
    skinTransform *= transform;
    skinTransform *= pivot;

    bonePivot *= Matrix.Invert(pivot);
    bonePivot *= transform;
    bonePivot *= pivot;

    _modelData.SetSkinTransform(boneName, skinTransform);
    _modelData.SetPivot(boneName, pivot);

    foreach(ModelBone child in bone.Children)
    {
        TransformBone(child.Name, pivot, transform);
    }
}
```

CODEVOORBEELD 10: BONE TRANSFORMATIE MET HIËRARCHIE.

Conclusie

Al met al zijn skinned animaties zeer goed te realiseren. Door de bone matrices goed te gebruiken zijn de pivots geen probleem. Verder moet alleen de shader aan het model worden gekoppeld om de bewegingen zichtbaar te maken. Uiteindelijk is deze techniek te gebruiken om bijvoorbeeld bewegingen van mensen of objecten na te bootsen op een realistische manier. Dat maakt het uitermate geschikt voor simulatie doeleinden. Wanneer je alle delen goed hebt toegepast, ben je in staat individuele bones binnen te transformeren en zichtbaar te maken. Dit kan worden gebruikt om bijvoorbeeld gamecharacters real time te animeren.

Kevin Lubbersen en Dennis Mijer, zijn aan het afstuderen aan Hogeschool Windesheim in Zwolle.