Praxisprojekt

Half-Edge Mesh für Unity3D

Erstellt von:
Yannick Dittmar
Studiengang: Allgemeine Informatik
Matrikelnummer: 11117676

Datum der Abgabe: xx.xx.xxxx

Betreuung: Dennis Buderus

Technische Hochschule Köln
Fakultät für Informatik und Ingenieurswissenschaften

1 Einleitung

In der Computergrafik werden dreidimensionale Modelle als Polygonen-Netz (Polygonal Mesh) dargestellt, um diese auf einem zweidimensionalen Bildschirm darzustellen. Die Oberfläche eines Modells wird dabei mit Hilfe von Polygonen angenähert. Häufig werden dafür Dreiecks-Netze (Triangle Mesh) verwendet, wobei die Flächen mit Dreiecken nachmodelliert werden, siehe Abbildung 1.
Ein solches Netz besteht aus den Eckpunkten der einzelnen Dreiecke, den Vertices. Diese werden durch Kanten (Edges) verbunden und bilden damit die Polygonalflächen, auch Faces genannt.

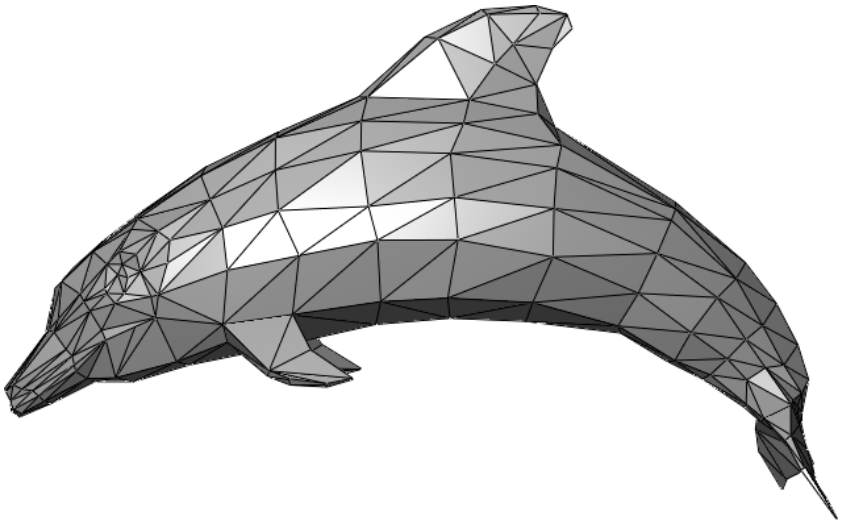

Abbildung 1: Beispiel eines Dreiecks-Polygonen-Netz, von [Wik]

2 Unity

Unity ist eine Spiele-Engine mit eingebauter Entwicklungsumgebung für 2D-, 3D- und VR-Spiele/-Simulationen. Die Engine kommt mit einem eigenen Editor, in welchem diverse Szenarien erstellt und bearbeitet werden können. Des Weiteren unterstützt Unity selbst programmierte Scripte auf der Grundlage von C#.

2.1 Meshes in Unity

Unity bietet die Möglichkeit, mit Hilfe von selbstgeschriebenen Scripten eigene 3D-Modelle zur Laufzeit erstellen zu lassen. Dafür stellt Unity ein eigenes Mesh-System zur Verfügung, basierend auf Dreiecksnetzen, die *UnityEngine.Mesh*-Klasse. Damit diese ein Mesh rendern kann, erwartet das Mesh ein *UnityEngine.Vector3*-Array für die Vertices, wobei ein *Vector3* einen Punkt im dreidimensionalen Raum darstellt und ein *int*-Array, das die Reihenfolge der Vertices für die Dreiecke festlegt.
Der folgende Code zeigt beispielhaft, wie ein Unity-Mesh erzeugt werden kann:

```
1  public void CreateMesh()
2  {
3    //--- Der Vollstaendigkeit halber vorhanden
4    meshFilter = gameObject.GetComponent<MeshFilter>();
5    if (meshFilter == null)
6      meshFilter = gameObject.AddComponent<MeshFilter>();
7
8    //--- vom MeshFilter zum Mesh
9    mesh = meshFilter.sharedMesh;
10   if (mesh == null)
11     mesh = new Mesh { name = "Quad" };
12
13   //--- MeshRenderer holen
14   meshRenderer = this.gameObject.GetComponent<MeshRenderer>();
15   if (meshRenderer == null)
16     meshRenderer = gameObject.AddComponent<MeshRenderer>();
17
18   //--- Mesh zusammenstellen
19   //--- Vertices/Points
20   var P0 = new Vector3(0, 0, 0);
21   var P1 = new Vector3(0, 1, 0);
22   var P2 = new Vector3(1, 0, 0);
23   var P3 = new Vector3(1, 1, 0);
24
25   var verticies = new List<Vector3> { P0, P1, P2, P3 };
26
27   //--- Triangles
28   var triangles = new List<int>();
29
30   triangles.Add(0);
31   triangles.Add(1);
32   triangles.Add(2);
33   triangles.Add(2);
34   triangles.Add(1);
35   triangles.Add(3);
36
37   //--- Mesh befuellen
38   mesh.Clear();
39   //--- Vertices zuweisen
40   mesh.vertices = verticies.ToArray();
41   //--- Triangles zuweisen
42   mesh.triangles = triangles.ToArray();
43   //--- Mesh dem MeshFilter zuweisen
44   meshFilter.sharedMesh = mesh;
45 }
```

Und liefert folgendes Ergebnis:

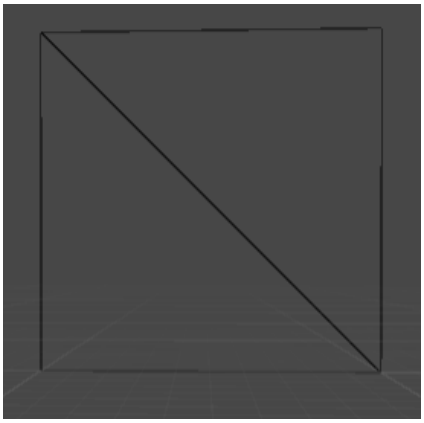

Abbildung 2: Die Wireframeansicht des erstellten Meshes im Unity Editor

2.2 Nachteile von Unity-Meshes

Die Vorteile bei dieser Herangehensweise sind, dass die von Unity verwendeten Netze auch bei größeren Netzen vergleichsweise wenig Speicherplatz benötigen, da dieser Ansatz auf eine direkte Repräsentation von Faces und Edges verzichtet. Daraus ergeben sich einige Nachteile. Durch die fehlende Referenz auf die Polygonenflächen sind Operationen auf Face-Ebene, wie eine Nachbarsuche, Abfragen auf alle Punkte Kanten oder die Unterteilung einzelner Polygonen in kleinere Einheiten (auch ,,Subdivision" genannt), sehr Zeitintensiv, weshalb sich solche Fälle nicht für Echtzeitanwendungen eignen. Des Weiteren stehen die Vertices im Unity-Mesh in keinem topographischen Zusammenhang, wodurch eine Iteration über das gesamte Netz erschwert wird, um beispielsweise zusammenhängende Punkte zu bearbeiten.

3 Half-Edge-Mesh

Um die oben genannten Problemstellungen zu lösen, gibt es andere Ansätze Polygonalnetze zu realisieren. Einer dieser Lösungen ist das Konzept der Half-Edge-Meshes. Ein solches Mesh besteht aus folgenden Komponenten:

- eine Liste von Vertices
- eine Liste von Faces
- eine Liste von Half-Edges.

Die Vertices sind, wie im Unity-Mesh auch, die Eckpunkte der Polygonen. Die Vertices werden mithilfe von Half-Edges miteinander verbunden.

3.1 Vertex

Ein Vertex ist ein Punkt im dreidimensionalen Raum und ist wie beim einfachen Mesh der Eckpunkt für die Polygonen. Neben einem *Vector3* für die Position des Vertex wird eine Referenz auf eine ausgehende Half-Edge, sowie der Index des Vertex benötigt.

```
1  public class Vertex
2  {
3    public event EventHandler PositionChangedEvent;
4
5    public Vertex(Vector3 point)
6    {
7      Point = point;
8      Index = -1;
9    }
10
11   public Vertex(Vector3 point, int index)
12   {
13     Point = point;
14     Index = index;
15   }
16
17   public Vertex(float x, float y, float z) : this(new Vector3(x, y, z))
18   { }
19
20   public Vertex(float x, float y, float z, int index) : this(new Vector3(x, y, z
21   { }
22
23   /// <summary>
24   /// The Index of the Vertex
25   /// </summary>
26   public int Index { get; set; }
27
28   private Vector3 _point;
29   /// <summary>
30   /// The Point
31   /// </summary>
32   public Vector3 Point
33   {
34     get => _point;
35     set
36     {
37       _point = value;
38       PositionChangedEvent?.Invoke(this, EventArgs.Empty);
39     }
40   }
```

```
41
42      /// <summary>
43      /// The Outgoing HalfEdge
44      /// </summary>
45      public HalfEdge HalfEdge { get; set; }
46  }
```

3.2 Face

Ein Objekt der Klasse Face repräsentiert die Faces des Meshs, die über die Vertices aufgespannt werden. Dabei besitzt eine Face nur die Referenz auf eine der anliegenden Half-Edges um das Traversieren und Bearbeiten des Netzes auf Face-Ebene zu ermöglichen. Die Face-Klasse ist wie folgt aufgebaut:

```
1   public class Face
2   {
3       public Face(HalfEdge halfEdge)
4       {
5           HalfEdge = halfEdge;
6       }
7
8       /// <summary>
9       /// A bonding HalfEdge
10      /// </summary>
11      public HalfEdge HalfEdge { get; set; }
12  }
```

3.3 Half-Edge

Die Half-Edge Klasse ist die wichtigste Komponente für das Half-Edge-Mesh. Sie stellt alle Teile in Beziehung zueinander. Zu bemerken ist, dass ein Half-Edge-Mesh keine explizite Modellierung von Edges hat. Eine Edge wird implizit durch zwei Half-Edges abgebildet. Jede Half-Edge besitzt eine Referenz auf ihre gegenüberliegende Half-Edge, sowie auf die ihr Nachfolgende. Zudem referenziert jede Half-Edge den Vertex, aus dem sie hervorgeht und die Face, die sie einschließt.

```
1   public class HalfEdge
2   {
3       public HalfEdge(Vertex outgoing, Face face, HalfEdge next)
4       {
5           OutgoingPoint = outgoing;
6           Face = face;
7           Next = next;
8           Index = -1;
9       }
```

```
10
11      public HalfEdge(Vertex outgoing, Face face, HalfEdge next, int index)
12      {
13          OutgoingPoint = outgoing;
14          Face = face;
15          Next = next;
16          Index = index;
17      }
18
19      public int Index { get; set; }
20
21      /// <summary>
22      /// The Vertex the HalfEdge comes from
23      /// </summary>
24      public Vertex OutgoingPoint { get; set; }
25
26      /// <summary>
27      /// The next HalfEdge, Counter Clockwise
28      /// </summary>
29      public HalfEdge Next { get; set; }
30
31      /// <summary>
32      /// The opposing HalfEdge
33      /// </summary>
34      public HalfEdge Opposing { get; set; }
35
36      /// <summary>
37      /// The face of the HalfEdge
38      /// </summary>
39      public Face Face { get; set; }
40
41      /// <summary>
42      /// Getter for the previous HalfEdge, for easier access
43      /// (this.Next.Next)
44      /// </summary>
45      public HalfEdge Previous => Next.Next;
46
47      /// <summary>
48      /// Getter for the EndPoint of this HalfEdge, for easier access (this.Next.Ou
49      /// </summary>
50      public Vertex EndPoint => Next.OutgoingPoint;
51  }
```

Literatur

[Wik] the free encyclopedia Wikipedia. Polygon mesh, 2007. [Online; zuletzt besucht am 04.12.2019].