Math2Model

Endbericht | Call 18 | Projekt ID 6890

Inhalt

1 Einleitung

Math2Model ist ein Werkzeug, das die Darstellung von mathematischen Funktionen in parametrischer Form in Echtzeit ermöglicht. Es erlaubt es somit, mathematische Funktionen im parametrischen Format zu rendern. Zwei Benutzeroberflächen werden unterstützt: Eine für Programmierer, in der diese mathematischen Objekte mittels Quellcode beschreiben können, und eine weitere Benutzeroberfläche für Nicht-Programmierer, die parametrisches Modellieren über einen Node-Editor ermöglicht.

2 Projektbeschreibung

Das Projektziel war, parametrische Modellierung in Echtzeit im Browser zu ermöglichen. Parametrische Modellierung ist ein gängiges Tool unter Architekten, 3D-Modellierern, Spieleentwicklern oder auch Künstlern, um schöne Formen über Mathematik zu erzeugen. Auch ein Einsatz unseres Tools in der Forschung ist denkbar. Ein einfach und v.A. schnell zu verwendendes Onlinetool in diesem Bereich gab es bislang noch nicht in dieser Form. Math2Model ist das erste derartige Tool, das Live-Modellierung über mathematische Formeln in Echtzeit im Browser ermöglicht, wobei sich „Echtzeit“ auf die Änderungen am Modell in Echtzeit – d.h. ohne merkliche Wartezeit – bezieht.

Typische Probleme von anderen Tools in dieser Hinsicht waren die Wartezeit auf die Ergebnisse (d.h. Ergebnisse waren nicht in Echtzeit sichtbar) oder die Erfordernis von (teils teuren) spezialisierten Tools. Modellierung im Browser war bis vor kurzem nur schwer möglich aufgrund von technischen Limitierungen.

Durch technischen Fortschritt in Browsern (genauer gesagt durch eine neue Programmierschnittstelle namens „WebGPU“) vor ca. 2 Jahren wurden neue technische Möglichkeiten geschaffen, die es erlauben, viel mächtigere Webapplikationen zu bauen. Und diese technische Weiterentwicklung ist auch Voraussetzung für unser Onlinetool, da der zugrundeliegende Algorithmus sogenannte „Compute Shader“ benötigt, die WebGPU nun auch im Web anbietet.

Die Ergebnisse unserer ca. 1,5-jährigen Arbeitszeit an diesem Projekt sind:

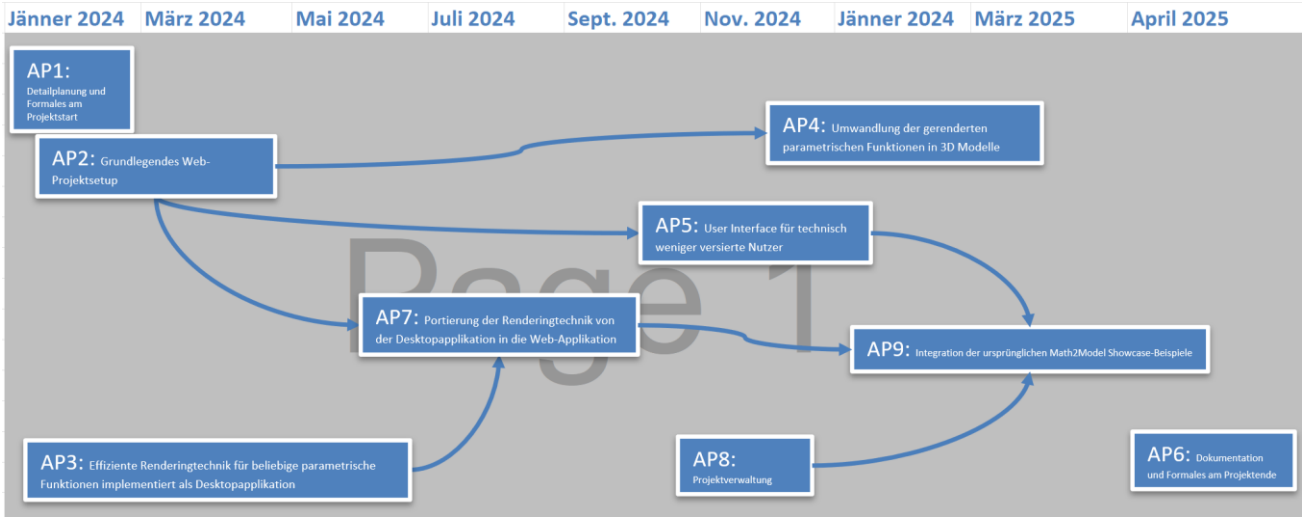
- Ein publiziertes Paper, das einen passenden Algorithmus beschreibt, um parametrisch definierte mathematische Modelle in Echtzeit mit hoher Framerate und hoher geometrischer Präzision zu rendern.
- Ein frei verfügbares Onlinetool, das den publizierten Algorithmus implementiert und folgende Features bietet:
    - Direkter Einstieg in die Modellierung über ein einfaches Webinterface

- Parametrische Modellierung über Sourcecode direkt im Browser für User, die mit Scripting oder Coding vertraut sind.
- Parametrische Modellierung über ein einfacheres Graphen-basiertes Interface, das diverse Hilfestellungen für technisch weniger versierte Nutzer bietet.
- Export der modellierten Formen in 3D Formate, um sie auch in anderen gängigen Programmen nutzen zu können (wie z.B. Spieleengines)

- Source Code und Dokumentation werden frei bereitgestellt.

3 Verlauf der Arbeitspakete

Das folgende Diagramm zeigt den aktualisierten Überblick über die Arbeitspakete und in welchen Monaten primär an diesen gearbeitet wurde:



3.1 **Arbeitspaket 1 -** ***Detailplanung und Formales am Projektstart***

Nötige Unterlagen zur Beantragung von Förderrate 1 wurden zusammengestellt und übermittelt. Im Zuge dessen haben wir uns auf ein neues Arbeitspaket geeinigt: AP9. Der Vertrag wurde unterschriegen, der Detailprojektplan erstellt und abgenommen.

Die Projektwebsite haben wir im Laufe der Entwicklungen schlussendlich auf GitHub Pages gehostet, was sehr gut mit unterschiedlicher Nutzerlast skalieren sollte. Ebenso konnten wir automatisches Deployment aus dem Coderepository dorthin einrichten.

Hier die Adresse der Website: https://math2model.cg.tuwien.ac.at/

3.2 Arbeitspaket 2 - ***Grundlegendes Projektsetup***

Die Haupttätigkeit dieses Arbeitspakets bestand darin, die technologische Basis unseres Onlinetools zu definieren. Dabei wurden eine Vielzahl von JavaScript-Frameworks ausprobiert.

Erkenntnisse wurden in folgenden beiden Blog Posts beschrieben:

- Entscheidungen hinsichtlich technologischer Basis
- WGSL - Die neue Shadersprache

Es gab dabei durchaus auch diverse Probleme – vor allem hinsichtlich des verwendeten Renderingframeworks Babylon.js. Es bietet eine Vielzahl an hilfreichen Funktionalitäten, doch wirft es auch Probleme auf, sodass wir immer wieder evaluieren mussten, ob es nicht zielführender wäre, Math2Model ohne ein großes Renderingframework zu entwickeln, sondern mittels direkter Verwendung der low-level API WebGPU.

Schlussendlich entschieden wir uns dann tatsächlich noch relativ spät – nämlich ziemlich genau in der Mitte der Projektlaufzeit – für einen Technologiewechsel auf direkte Verwendung der WebGPU-Schnittstelle, die wir einerseits über TypeScript – aber andererseits auch über die Programmiersprache Rust bedienen können. Diese Vorgehensweise (sowohl TypeScript als auch Rust) hat Vor- und Nachteile. Zu den Vorteilen zählen die hohe Flexibilität für Entwickler sowie die Möglichkeit, GPU-Profiling-Tools verwenden zu können – als Nachteile ist vor Allem die erhöhte Komplexität des Projekts aus Entwicklersicht sowie das leicht kompliziertere Projektsetup zu nennen.

Rust ist eigentlich eine Desktopprogrammiersprache, die aber mittels WebAssembly in ein Browserproramm übersetzt und somit dafür genutzt werden kann. Weitere Details haben wir in diesem Blogpost beschrieben:

- Browserprogrammierung mit einer "Desktop-Programmiersprache"

3.3 Arbeitspaket 3 - *Effiziente Renderingtechnik für beliebige parametrische Funktionen*

AP3 bezieht sich nun rein auf die Entwicklung einer effizienten Renderingtechnik für parametrisch definierte Objekte. Dieses wurde zunächst als eigenständige Desktopapplikation entwickelt, getestet, evaluiert und bei einer internationalen Konferenz eingereicht. Verwandt dazu ist AP7: Es bezieht sich darauf, die Renderingmethode in unser Onlinetool zu portieren und dort verwendbar zu machen.

Unsere Vorgehensweise, die Technik zuerst als Desktopapplikation zu entwickeln hat den großen Vorteil, dass wir alle Funktionalitäten von Graphikkarten nutzen konnten, um die effizienteste Variante der Renderingtechnik zu entwickeln. Die Browserbasierte WebGPU-Schnittstelle ist hinsichtlich verfügbarer Funktionalitäten etwas limitiert, was daran liegt, dass sie höchstmögliche Kompatibilität für Web-fähige Geräte ermöglichen will.

Wir konnten AP3 vollständig abschließen. Unsere effiziente Renderingmethode wurde von internationalen Reviewern begutachtet, zum The Eurographics Symposium on Parallel Graphics and Visualization (EGPGV) zugelassen, dort präsentiert und mit dem Best Paper Award ausgezeichnet. Das Paper wurde bereits von The Eurographics Association publiziert.

Weitere Details zur verwendeten Renderingtechnik sind in diesem Blogpost beschrieben:

- Publikation unserer Renderingtechnik

3.4 **Arbeitspaket 4 -** *Umwandlung der gerenderten parametrischen Funktionen in 3D Modelle*

Beschreibung der Geometrie eines Objektes über Mathematik ist zwar eine sehr elegante Form und auch sehr speichereffizient, da eben nur die mathematische Formel benötigt wird – allerdings ist diese Beschreibung der Geometrie auch mitunter unpraktisch, da sie von klassischen Graphiktools nicht unterstütz wird.

Während unser Tool die mathematischen Funktionen direkt darstellen kann, benötigt es mitunter andere Formate, um die so erzeugte Geometrie auch in anderen Tools verfügbar zu machen. Klassisischerweise wurde und wird 3D-Geometrie über 3D-Modelle ausgetauscht. Dafür gibt es eine vielzahl an gängigen Formaten. Zwei der gängigsten Formate sind OBJ und glTF. Wir haben uns dafür entschieden, diese beiden zu unterstützen, da sie sehr gängig und offene Standards sind.

Die Herangehensweise beim Exportieren der 3D-Modelle ist eine etwas andere als bei der Darstellung am Bildschirm, das eine Änderung am grundlegenden Algoithmus erfordert: Anstatt den Detailgrad der Geometrie adaptiv zum Bildausschnitt anzupassen, wird der Detailgrad uniform gesetzt und so exportiert.

Herausforderungen bei diesem Schritt waren die Auswahl von passenden 3D-Formaten und Implementierung des Exports, sowie das Finden passender Export-Einstellungen. Schlussendlich haben wir einige Exporteinstellungen konfigurierbar gemacht, sodass User diese an ihre Anforderungen anpassen können.

In Summe hat die Entwicklung deutlich länger gedauert als geplant – vor Allem wegen dem Finden passender Exporteinstellungen, Verhindern dass exportierte Meshes nicht mehr zusammenhängen und auch wegen technischen Limitierungen hinsichtlich WebGPU, welches Beschränkungen für die maximale Buffergröße mit sich bringt. Schlussendlich konnten wir

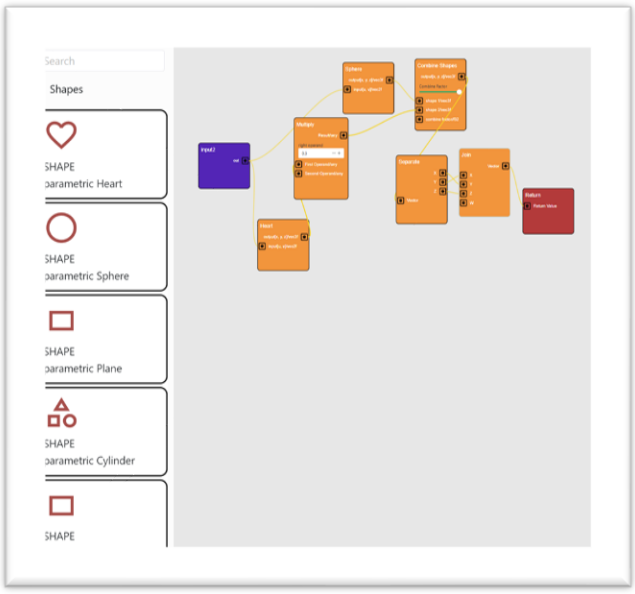
3.5 **Arbeitspaket 5 -** *User Interface für technisch nicht versierte Nutzer*

Dieses Paket war ein ein besonders wichtiges Arbeitspaket, da wir für unterschiedliche Usergruppen ein geeignetes Interface anbieten wollten. Einerseits erlaubt unser Tool, direkt Quellcode einzugeben – wie bei unserem großen Vorbild, der Website Shadertoy.com – was gut geeignet ist für versierte User; aber wir wollten auch für technisch weniger versierte User eine Möglichkeit implementieren, unser Tool nutzen zu können. Für diesen Zweck haben wir einen Node Editor implementiert, der Modellierung ohne Programmierkenntnisse ermöglicht.

Das untenstehende Bild zeigt beide Arten der Eingabe: Links das Programmierinterface und rechts der Node Editor, der keine Programmierkenntnisse erfordert – sehr wohl allerdings ein

gewisses mathematisches Verständnis, aber das liegt generell an der Art der Modellierung über parametrische Funktionen, auf die unser Tool abzielt.

```
{
  return x - floor(x * (1.0 / 289.0)) * 289.0;
}

fn permute(x: vec4f) -> vec4f
{
  return mod289(((x*34.0)+10.0)*x);
}

fn taylorInvSqrt(r: vec4f) -> vec4f
{
  return 1.79284291400159 - 0.85373472095314 * r;
}

fn fade(t: vec2f) -> vec2f {
  return t*t*t*(t*(t*6.0-15.0)+10.0);
}

fn cnoise(P: vec2f) -> f32
{
  var Pi = floor(P.xyxy) + vec4f(0.0, 0.0, 1.0, 1.0);
  var Pf = fract(P.xyxy) - vec4f(0.0, 0.0, 1.0, 1.0);
  Pi = mod289(Pi);
  var ix = Pi.xzxz;
  var iy = Pi.yyww;
  var fx = Pf.xzxz;
  var fy = Pf.yyww;

  var i = permute(permute(ix) + iy);

  var gx = fract(i * (1.0 / 41.0)) * 2.0 - 1.0 ;
  var gy = abs(gx) - 0.5 ;
  var tx = floor(gx + 0.5);
  gx = gx - tx;

  var g00 = vec2f(gx.x,gy.x);
  var g10 = vec2f(gx.y,gy.y);
```



Bei der Implementierung dieses Arbeitspakets gab es zwei große Herausforderungen:

1. Das Finden eines geegneten Frameworks für den Node Editor. Alternativ haben wir eine eigene Implementierung angedacht – doch schlussendlich konnten wir doch das JavaScript Framework Rete.js zuhilfe nehmen.
2. Die Wahl der Abstraktion, die die Knoten in unserem Node Editor bieten, war eine Herausforderung für sich. Unser Tool bietet hier sowohl Knoten für Mathematische Funktionen wie z.B. Sinus oder Cosinus – aber wir wollten hier noch etwas mehr Komfort bieten und das führte schlussendlich dazu, dass wir auch Knoten im User Interface anbieten, die für sich genommen schon parametrische Funktionen sind. Ein Knoten ist z.B. ein parametrisches Herz. Überdies gibt es diverse geometrische Grundformen wie z.B. eine Kugel. User können diese als Basis nehmen und nach eigenen Vorstellungen noch bearbeiten. Gleichzeitig bietet der Node Editor nicht die volle Funktionalität der Code Editors – doch in Summe dürfte unsere Wahl der Abstraktion im Node Editor tatsächlich dazu führen, dass sich User mit unterschiedlichen Fähigkeiten und Spezialisierungen im Tool zurechtfinden.

Weitere Details über das User-Interface für technisch weniger versierte User finden sich in diesem Blogpost:

- Einfaches User-Interface (ohne Programmierkenntnisse)

Die beiden unterschiedlichen Eingabemethoden werden noch von einem unserer studentischen Mitarbeiter im Zuge einer Bachelorarbeit mittels User Study nach netidee-Projektabschluss

evaluiert werden. Die Resultate können wir gerne bereitstellen und in einem weiteren Blog Post zusammenfassen.

3.6 **Arbeitspaket 6 -** ***Dokumentation und Formales am Projektende***

Wir haben zweierlei Dokumentation erstellt:

- Die Dokumentation für Entwickler ist direkt beim Source Code auf GitHub zu finden, wo das Projekt unter https://github.com/cg-tuwien/Math2Model zu finden ist. Neben der Dokumentation in der README.md-Datei, stellen wir noch weitere ausgewählte relevante Detailinformationen im /docs-Verzeichnis bereit.
- Die Dokumentation für User ist direkt über unser Tool (welches unter https://cg-tuwien.github.io/Math2Model/ erreichbar ist) aufrufbar. Der Menüpunkt *Help -> Documentation* führt zur Dokumentation und bietet diverse Hilfeseiten zu den einzelnen Teilen unseres Onlinetools.

Die weiteren erforderlichen Unterlagen haben wir direkt an netidee übermittelt.

Den Projektendbericht werden wir über die netidee-Projektwebsite zugänglich machen: https://www.netidee.at/math2model

3.7 **Arbeitspaket 7 -** ***Portierung der Renderingtechnik von der Desktopapplikation in die Web-Applikation***

Die Renderingtechnik, die in AP3 implementiert und publiziert wurde, wurde in die Webapplikation portiert. Dabei wurde Rücksicht auf besondere Vorgaben und Einschränkungen von Webapplikationen genommen und entsprechende Anpassungen vorgenommen. Während die publizierte Renderingtechnik auf Desktop-Graphikschnittstellen entwickelt wurde, ist Math2Model ein Onlinetool, das eine Web-Graphikschnittstelle (nämlich WebGPU) verwendet. Letztere hat einige Einschränkungen im Vergleich zu ersterer, daher war es nötig, unseren Algorithmus etwas anzupassen, um ihn im Web lauffähig und performant zu machen:

- Für WebGPU müssen wir dessen Shadersprache WGSL verwenden, die im Vergleich zu GLSL für Vulkan ein paar Einschränkungen hinsichtlich der sog. Subgroup-Operationen hat. Stattdessen verwenden wir workgroup-local memory und memory barriers.
- Unser Onlinetool unterstützt nur die "tessellated patches" Renderingstrategie, wobei „software rasterization“ aufgrund von technischen Einschränkungen nicht sinnvoll umsetzbar gewesen wäre (kein Support für atomic operations auf 64-bit Datentypen)
- Da Tessellation Shader in WebGPU nicht unterstützt sind, verwenden wir eine Art „manuelle Tessellation“: Wir rendern vortessellierte Quads fixer Größen (2x2, 4x4, 8x8, ..., 32x32))

- Für die Patch Subdivision benötigen wir einen Zwischenschritt um auszurechnen, wie viele workgroups wir dispatchen müssen. Wenn wir mehr als 65535 patches in einem subdivision Schritt haben, dann müssen die compute workgroups aufgeteilt werden.
- Es gibt einen extra Schritt vor dem Rendering, um die indirect draw counts fürs das Rendering zu kopieren.
- Die Patches werden in einem Speicher-effizientem Encoding abgespeichert {u: u32, v: u32} statt { min: vec2, max: vec2 }.

Weitere Implementierungsdetails beschreiben wir in folgendem Blogpost:

- Browserprogrammierung mit einer "Desktop-Programmiersprache"

3.8 **Arbeitspaket 8 –** ***Projektverwaltung***

Eine Projektverwaltung wurde implementiert werden, sodass User ihre Projekte speichern, laden und teilen können. Es ist nicht nötig, Nutzerkonten anzulegen - das hat zwei Vorteile: Einerseits kann die Projektverwaltung von Nutzern ohne jegliche Hürden verwendet werden. Andererseits ist kein serverseitiges Scripting nötig, was den Entwicklungsaufwand und die Projektkosten im Rahmen hält.

Ein Projekt besteht aus diversen Dateien, die in einem ZIP-Archiv gesammelt gespeichert oder von einem geladen werden können. Durch diese Vorgehensweise ist die Projektverwaltung einfach und transparent ohne Verwendung irgendwelcher proprietären Dateiformate.

Projekte können gespeichert werden über den Menüpunkt *File -> Save As*.
Laden kann man ein Projekt analog dazu über *File -> Open*.

3.9 **Arbeitspaket 9 -** ***Integration der ursprünglichen Math2Model Showcase-Beispiele***

Beim Projektantrag haben wir ein kleines Showcase erstellt, wie das Tool aussehen könnte. Enthalten waren vier Beispielprojekte. Diese Beispielprojekte haben wir in angepasster Form auch in unser Endprojekt übernommen. Sie können geladen werden über *File -> Examples*.

Dabei ist wichtig zu beachten, dass dei Beispiele teils etwas anders aussehen als im Showcase. Das liegt einerseits daran, dass sie nun in ein allgemeines Modellierungstool integriert worden sind, das einen gewissen Rahmen vorgibt. Das Showcase war pro Beispiel händisch adaptiert.

Einige Teile der Beispiele haben wir mittels Code Editor umgesetzt – andere über den Node Editor, um beide Modellierungsoptionen zu veranschaulichen und Vorlagen für User zu bieten.

Teils war es nicht ganz einfach die Beispiele zu portieren, doch konnten wir gute Kompromisse finden. So war etwa das „Turm“-Exam nicht 1:1 portierbar, weil es im Showcase ein bestehendes

3D-Modell geladen und dieses verändert hat. Das Pendant in unserem Endprodukt beschreibt ein komplettes Turm-Modell auf parametrische Weise.

4 Umsetzung Förderauflagen

Unsere Förderauflagen besagen, dass wir einen Teil des Projektes selbst zu finanzieren haben. Die interne Finanzierung von Projektmitarbeitern und Studenten wurde sichergestellt.

5 Liste Projektendergebnisse

1	**Projektzwischenbericht**	CC BY 4.0	https://www.netidee.at/math2model
2	**Projektendbericht**	CC BY 4.0	https://www.netidee.at/math2model
3	**Entwickler_innen-DOKUMENTATION** des Projektergebnisses für andere Entwickler_innen ("Dritte"), die das Projektergebnis nach Projektende nutzen/weiterentwickeln wollen	CC BY 4.0	https://github.com/cg-tuwien/Math2Model/blob/main/README.md
4	**Anwender_innen-DOKUMENTATION** des Projektergebnisses für Anwender_innen, die das Projektergebnis nach Projektende nutzen wollen	CC BY 4.0	https://cg-tuwien.github.io/Math2Model/docs/
5	**Veröffentlichungsfähiger Einseiter / Zusammenfassung**	CC BY 4.0	https://www.netidee.at/math2model
6	Dokumentation **Externkommunikation** zur Erreichung Sichtbarkeit /Nachhaltigkeit: als Teil des Endberichtes	CC BY 4.0	https://www.netidee.at/math2model
7	**Rendering von parametrischen Modellen mittels Code und deren Rendering** - Online Codeeditor (ähnlich zu Shadertoy) - Rendering von beliebigen, durch parametrische Funktionen beschreibbaren, Objekten - Live-Updates der parametrischen Modelle - Rendering in Echtzeit (dank WebGPU und dessen Compute Shader-Support)	MIT	https://math2model.cg.tuwien.ac.at/
8	**Generierung von parametrischen Modellen mittels einsteigerfreundlichem User Interface** - Einsteigerfreundlicher online Formeleditor - Auswahl aus vorgefertigten Elementen (z.B. parametrische Kugel, parametrisches Herz, etc.) - Änderung der jeweiligen Parameter über ein einfaches User Interface	MIT	https://math2model.cg.tuwien.ac.at/

9	**Generierung von parametrischen Modellen mittels Graph** - Graphenbasierte Modellierung von parametrischen Funktionen (ähnlich wie Code, nur über Graphen, wie sie mitunter auch in anderen professionellen Werkzeugen zum Einsatz kommen)	MIT	https://math2model.cg.tuwien.ac.at/
10	**Export der parmetrischen Modelle in ein Dreiecksmodell** - Online erstelltes parametrisches Modell kann exportiert werden - Zwei gängige 3D Formate werden unterstützt	MIT	https://math2model.cg.tuwien.ac.at/
11	**Projektverwaltung** - Projekte können kopiert, gespeichert und geladen werden - Projekte können über eine ZIP-Datei geshared werden	MIT	https://math2model.cg.tuwien.ac.at/
12	**Beispiele der ursprünglichen Math2Model Showcase Applikation in das neue WebGPU-basierte Onlinetool integrieren** - Die ursprünglichen Showcase-Beispiele werden als Examples in das neu erstellte Onlinetool integriert - Die Showcase-Beispiele können von Nutzern zu Lernzwecken genutzt werden.	MIT	https://math2model.cg.tuwien.ac.at/

6 Verwertung der Projektergebnisse in der Praxis

Die Projektergebnisse können in der Praxis wie folgt verwendet werden:

- Unkomplizierte, spontane parametrische Modellierung über unsere Website https://cg-tuwien.github.io/Math2Model/
- Analyse der beiden unterschiedlichen Modellierungsmethoden im Zuge einer Bachelorarbeit. Im Zuge der Evaluierung werden die Teilnehmer einer User Study unser Tool benutzen und dessen beide unterschiedliche Modellierungsvarianten (Source Code und Node Editor)
- Die Anpassungen, die in Kapitel 3.7 Arbeitspaket 7 - Portierung der Renderingtechnik von der Desktopapplikation in die Web-Applikation beschrieben wurden sind interessant für die WebGPU-Community und können als wertvolle Richtlinien bzw. als Feedback für andere WebGPU-Projekte dienen.

7 Öffentlichkeitsarbeit/ Vernetzung

Die Renderingtechnik wurde wie unter 3.3 Arbeitspaket 3 beschrieben publiziert und auf der EGPGV Konferenz präsentiert.

Weitere Sichtbarkeit wird unser Tool durch zwei Bachelorarbeiten und eine Diplomarbeit unserer studentischen Mitarbeiter erlangen:

- Eine Bachelorarbeit wird die beiden unterschiedlichen Modellierungsvarianten evaluieren (Programmierinterface vs. Node Editor) für User mit unterschiedlichen technischen Hintergründen.
- Eine Bachelorarbeit wird den 3D-Modellexport genauer beschreiben, sowie die damit verbundenen Optionen, Probleme und Herausforderungen.
- Die Diplomarbeit wird ein nicht direkt mit Math2Model verbundenes Thema behandeln, doch durch eine Zitation ist auch hier mehr Sichtbarkeit für Math2Model gegeben.

8 Eigene Projektwebsite

Unser Tool ist online verfügbar auf der eigens eingerichteten Website:
https://cg-tuwien.github.io/Math2Model/

Dort sind sämtliche Funktionen verfügbar, inklusive Projektverwaltung, Beispielen und Dokumentation.

9 Geplante Aktivitäten nach netidee-Projektende

Ja, direkt im Anschluss an das netidee-Projektende sind folgende weitere Aktivitäten geplant:

- Evaluierung der beiden unterschiedlichen Modellierungsoptionen (Code Editor und Node Editor) in einer User Study im Kontext einer Bachelorarbeit. Die Zusammenfassung der Ergebnise werden wir als Blog Post auf der netidee-Projektwebsite veröffentlichen.
- Beschreibung der Optionen, Probleme, Herausforderungen und Eigenheiten bei der Umwandlung von parametrischen Modellen in 3D-Modelle und beim Export derer im Zuge einer weiteren Bachelorarbeit. Eine Zusammenfassung der Erkenntnisse werden wir als Blog Post auf der netidee-Projektwebsite veröffentlichen.

10 Anregungen für Weiterentwicklungen durch Dritte

Es gibt diverse Anregungen für Dritte, wie das Tool verbessert werden könnte. Hier eine kleine Auswahl:

- Implementierung weiterer Nodes im Node Editor, sodass noch mehr Bausteine out-of-the-box geboten werden.

- Weitere Optionen für die Texturierung von parametrischen Objekten, wie z.B. Support für mehrere Texturen (und Blending zwischen ihnen) für ein Objekt
- Verbesserung der Renderingperformance in großen Szenen: Die LOD Stage aus unserer Renderingtechnik könnte so umgebaut werden, dass sie nicht pro Objekt separat aufgerufen wird, sondern nur ein mal – und die so aufgerufene Compute Shader-Hierarchie würde alle Objekte parallel analysieren (ähnlich wie in der publizierten Renderingtechnik – siehe 3.7 Arbeitspaket 7 - Portierung der Renderingtechnik von der Desktopapplikation in die Web-Applikation).
- An diversen Stellen könnte die Nutzeroberfläche noch etwas gepolished werden und die Usability verbessert werden – wie z.B. durch Verwendung von kontrastreicheren Farben, sodass Menschen mit Einschränkungen in der Farbwahrnehmung besser berücksichtigt werden.

Dabei gilt es stets zu bedenken, wo sich der jeweilige Code befindet: Das Rendering Backend ist in Rust implementiert, um auch effizientes Debugging als Desktopapplikation mittels Profiling-Tools zu ermöglichen; die meisten anderen Komponenten sind in TypeScript implementiert, wie z.B. der Node Editor. Über eine eigens Implementierte Schnittstelle kommunizieren Rust und TypeScript.

11 Externkommunikation

Für externe Kommunikationsmöglichkeiten haben wir entsprechende Informationen und Links auf der netidee-Projektseite bereitgestellt:

- Link zu den Issues unseres GitHub Repositories für Fragen bzgl. Entwicklung
- Link zum Impressum unserer Forschungsgruppenwebsite für allgemeine/organisatorische Fragen

Erhöhung der Sichtbarkeit:

- Verwendung des Tools im Kontext von Bachelorarbeiten: Der Aufwand ist hier hauptsächlich in der Erstellung der Arbeit, während der Nutzen erhöhte Sichtbarkeit durch Verwendung und Zitation ist.
- News-Eintrag auf unserer Forschungsgruppenwebsite https://www.cg.tuwien.ac.at/ - Nach Projektabnahme werden wir ein entsprechendes Newsitem erstellen. Unsere Website hat durchaus eine signifikante Reichweite sowohl unter Studierenden als auch unter externen Forschern.

Lessons Learned:

- Die größten Probleme in unserem Tool waren technischer Natur:
    - Inkompatibilitäten zwischen Frameworks
    - Schlechte bis unmöglches Debugging von TypeScript-Programmen. Die Lösung war in unserem Fall die Verwendung von Rust; aber das bringt eigene Probleme mit sich:
        - Translationslayer zwischen den beiden „Welten“ Rust und TypeScript
- Eine weitere Befürchtung unsererseits ist, dass der Node Editor, der für technisch weniger versierte Nutzer gedacht ist, trotzdem ein gewisses technisches Verständnis verlangt, der über das rein mathematische Verständnis hinausgeht – weitere Erkenntnisse diebezüglich werden wir im Zuge einer Bachelorarbeit sammeln, die eine entsprechende User Study durchführen wird.
    - Basierend auf den Ergebnissen der User Study, sind noch diverse kleinere Optimierungen denkbar
    - Generell gilt ein Node Editor als die Standardherangehensweise in vergleichbaren Tools – für zukünftige Weiterentwicklungen wären ggf. auch komplett andere Herangehensweisen interessant, falls solche gefunden werden können.