# Project 4: Intro to Interpolation, Tessellation & Geometry Shaders

Publish | Edit | ⋮



**GPR-300 Intermediate Graphics & Animation Programming**
**Instructor: Daniel S. Buckstein**
**Project 4: Intro to Interpolation, Tessellation & Geometry Shaders**

**Summary:**
So far we have explored mostly different lighting and shading techniques involving forward and deferred pipelines. Now we will change gears and focus on the new types of shaders and their applications, as well as improving the visual fidelity of the effects we have already implemented.

**Objectives:**
Upon successful completion of this assignment, you will have accomplished the following:

- Revisit and implement an updated forward shading pipeline, adding parallax occlusion mapping (POM) for fake level-of-detail.
- Explore tessellation shaders for geometric level-of-detail and interpolation.
- Explore geometry shaders for debugging overlays.

**Submission:**
Start your work immediately by ensuring your coursework repository is set up, and public. Create a new main branch for this assignment. ***Please work in pairs (see team sign-up). Begin your submission immediately, copy the following into the text box, decide on your repository branch name for this assignment and fill in the information below.*** **The submission box locks at the specified deadline; be proactive and don't miss it. Late submissions, even by a minute, will not be accepted.**

***Copy, edit and submit the following text once as a team (you do not need the headings, please just provide your info as shown in the examples here)***:

1. ***Names of contributors***: Write the names of the contributors of this assignment.
   e.g. **Dan Buckstein**
2. ***A link to your public repository online***: Grab the clone link from your working repository, which should end with "*.git*".
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework.git** (note: not a real link)
3. ***The name of the branch that will hold the completed assignment***: Create a new branch for this project, submit only the name of this branch.
   e.g. **project0-main**
4. ***A link to the read-me and user instructions for this project***: Ensure your repository includes a read-me that summarizes how to use your finished product.
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework/blob/project0-main/project0-readme.pdf** (note: not a real link)
5. ***A link to your video (see below) that can be viewed in a web browser***: Ensure your video is public or shared with your instructor.
   e.g. YouTube link: **https://www.youtube.com/watch?v=OqOyxQVs8lY** (note: "Attack on Game Development" by Will Gordon & Connor Breen)

Finally, please submit a ***10-minute max*** demo video of your project. Use the screen and audio capture software of your choice, e.g. Google Meet, to capture a demo of your project as if it were in-class. This should include at least the following, in enough detail to give a thorough idea of what you have created (hint: this is something you could potentially send to an employer so definitely show off your professionalism and don't minimize it):

- Show the final result of the project and any features implemented, with a voice over explaining what the user is doing.
- Show and explain any relevant contributions implemented in code and explain their purpose in the context of the assignment and course.
- Show and explain any systems source code implemented, i.e. in framework or application, and explain the purpose of the systems; this includes changes to existing source.
- **DO NOT AIM FOR PERFECTION, JUST GET THE POINT ACROSS**. Please mind the assignment rubric to make sure you have demonstrated enough to cover each category.
- **Please submit a link to a video visible in a web browser, e.g. YouTube or Google Drive.**
- **Instead of waiting until last-minute for the video to upload, create a folder or links document on Drive that is accessible to your instructor. Submit the link to this folder as part of your official submission, and copy your video file/link there when it is ready.**

**Instructions & Requirements:**

***DO NOT begin programming until you have read through the complete instructions, bonus opportunities and standards, start to finish. Take notes and identify questions during this time. The only exception to this is whatever we do in class.***

Using the provided framework, implement the following:

1. **Setup framework**: Complete the following steps to ready the framework for this project:
   A. ***Project branch***: Check out the project starter branch "*graphics2/proj4*" to begin the project.
   B. ***Implement renderer utilities***: Here are the steps for setting up this demo:
      I. *Explore pre-built example*: Load the pre-built example demo for this project: "*File > Load demo > ...Proj3*". The demo shows the completed scene with procedural and loaded models, and the ability to toggle a variety of pipeline stages and features.
      II. *Explore renderer library*: The renderer library source is now available. Browse through the source for the "*animal3D-A3DG-OpenGL*" library.
         - Take a look at the header and interface for the drawable object, namely "*a3_VertexDrawable-OpenGL*" to better understand how models are drawn using OpenGL and the animal3D a3_VertexDrawable interface.
      III. *Initial build and run*: Either build the project in Visual Studio then launch using "*File > DEBUG... > Load without building*", or use "*Quick build*" to hot-build and run the demo directly through the player window.
      IV. *Additional setup*: Complete the following steps to make sure the framework is all set:
         - Navigate to the intro demo mode filter: "*Source Files/common/A3_DEMO/a3_DemoMode3_Curves*"; open the update source for this mode: "*a3_DemoMode3_Curves-idle-update.c*". Here you must update the movable object's position so that it follows the curve path.
         - Navigate to the render source for this mode: "*a3_DemoMode3_Curves-idle-render.c*". Here you must uncomment the existing rendering pipeline elements and complete them to have a functional render pipeline.
         - You will want to do some reading about tessellation to better understand how to implement the missing parts:
            - **https://www.khronos.org/opengl/wiki/Tessellation (https://www.khronos.org/opengl/wiki/Tessellation)**
            - **https://www.khronos.org/opengl/wiki/Tessellation_Control_Shader (https://www.khronos.org/opengl/wiki/Tessellation_Control_Shader)**
            - **https://www.khronos.org/opengl/wiki/Tessellation_Evaluation_Shader (https://www.khronos.org/opengl/wiki/Tessellation_Evaluation_Shader)**
2. **Implement shaders**: Using the pipeline implemented in part 1:
   A. ***Encode/decode shaders***: Upon successful completion of the render pipeline, using the encoded shaders should allow you to traverse the complete set of passes to see

how the bloom effect is achieved. For your actual implementation, remove the "*e/*" from each shader file path.
- With encoded shaders disabled (required), the first two modes (NM and POM) are identical, and the final mode shows empty space.
- Since the last project, a bunch of the decoded shaders have been fully or partially implemented.

B. ***Implement shader programs***: Implement the following effects by visiting and completing the following GLSL files (navigate to "*Resource Files/A3_DEMO/glsl/4x*"):
   I. *Curve interpolation*: The first task is to visualize the path of the animated object. We have a curve made up of segments, each with a predetermined "waypoint" representing the ends. In this approach, we draw a simple line segment between two waypoints (an "isoline patch"), and use *tessellation* to subdivide the segment into smaller pieces. While tessellating, we also perform spline interpolation to produce a curve.
      - Code setup: In "*a3_DemoMode3...render.c*", implement the function "*a3vertexDrawableRenderIsoPatches*".
      - Vertex shader: This program uses the vertex shader "*vs/03-lod/empty_vs4x.glsl*" which is a vertex shader that does literally nothing. No attributes, no transformations, nada. We will have our tessellation shader generate and transform the vertices.
      - Tessellation control shader: Open "*ts/03-lod/tessIso_tcs4x.glsl*" and complete the to-do prompts. This shader simply specifies how many sub-segments into which we want our line segment divided; the more divisions, the finer the curve.
      - Tessellation evaluation shader: Open "*ts/03-lod/passColor_interp_tes4x.glsl*" and complete the to-do prompts. This shader evaluates the vertices generated from tessellation; our job is to call the appropriate spline interpolation algorithm and set the clip-space position, passing out whatever color we want.
   II. *Phong shading with parallax occlusion mapping*: In this project, we are fundamentally exploring "level-of-detail" techniques to improve visual fidelity of our models. While normal mapping helps emphasize the effect of light across the surface, *parallax occlusion mapping* (POM) will create the illusion of depth and ridges across a surface. See Blue Book for examples.
      - Vertex shader: This program uses "*vs/02-pipeline-deferred/passTangentBasis_ubo_transform_vs4x.glsl*" which has been implemented for your convenience.
      - Fragment shader: Open "*fs/03-lod/drawPhongPOM_fs4x.glsl*" and follow the to-do prompts. To start off, it is complete to the point of producing the normal mapping effect; our job is to implement the POM algorithm and invoke it correctly.

III. *Tangent bases*: While the models are rendered with proper lighting above, it is a good idea to have some debugging visualizations. Here we implement an alternative drawing pipeline that reads model triangles and converts them into lines, so we can output things like wireframe or the tangent basis at each vertex. This effect is achieved using a *geometry shader*.
   - Vertex shader: This program uses the same vertex shader as above (POM).
   - Geometry shader: Open "*gs/00-common/drawTangentBases_fs4x.glsl*" and follow the to-do prompts. This shader reads entire triangles (3 vertices) and outputs vertices to construct lines.

IV. *Level-of-detail*: While POM is a per-fragment technique for producing the illusion of detail, how does that compare with changing the geometry to adjust the actual vertices? Using triangular patches, we can tessellate our models and adjust the vertices using displacement (height map).
   - Code setup: In "*a3_DemoMode3...render.c*", implement the function "*a3vertexDrawableRenderTriPatches*".
   - Vertex shader: This program uses the same vertex shader as above (POM).
   - Tessellation control shader: Open "*ts/03-lod/tessTriTangentBasis_tcs4x.glsl*" and follow the to-do prompts. This shader defines the tessellation pattern for the incoming triangles, and passes along the vertex data for lighting (we are ultimately using the normal mapping effect).
   - Tessellation evaluation shader: Open "*ts/03-lod/passTangentBasis_displace_tes4x.glsl*" and follow the to-do prompts. This shader interpolates the tessellated vertices into place, and performs the displacement, re-transforming the vertex into clip space and passing along the lighting data.

3. **Testing & demonstration**: You must thoroughly test your render pipeline and shaders. Your demonstration must show evidence of the following:
   A. **Walkthrough and justification of architecture**: Demonstrate that the above requirements have been met in code.
   B. **Framework**: Demonstrate completion of the pipeline setup using animal3D data structures and functions.
   C. **Shaders**: Demonstrate the functional rendering pipeline and completion of the shaders implemented throughout the project. Furthermore, you must demonstrate that the shaders are your own and not the encoded files.
   D. **Takeaways**: Discuss personal and professional takeaways from this project.

**Bonus:**
You are encouraged to complete one or more of the following bonus opportunities (rewards listed):

- ***Renderer extension (+2)***: The special patch rendering functions are specific to the demo mode for this project. Formally move them into the renderer project source and

integrate them with the existing interfaces. This will allow patch rendering for tessellation to happen anywhere at any time. This will take more effort than a simple copy-paste.
- ***New tessellation effect (+2)***: Create a brand new line or triangle tessellation effect, such as a geometric fractal.
- ***New geometry effect (+2)***: Create a brand new effect using a geometry shader, such as exploding a model.

**Coding Standards:**
You are required to mind the following standards (penalties listed):

- ***Reminder: You may be referencing others' ideas and borrowing their code. Credit sources and provide a links wherever code is borrowed, and credit your instructor for the starter framework, even if it is adapted, modified or reworked (failure to cite sources and claiming source materials as one's own results in an instant final grade of F in the course)***. Recall that borrowed material, even when cited, is not your own and will not be counted for grades; therefore you must ensure that your assignment includes some of your own contributions and substantial modification from what is provided. ***This principle applies to all evaluations***.
- ***Reminder: You must use version control consistently (zero on organization)***. Commit after a small change set (e.g. completing a section in the book) and push to your repository once in a while. Use branches to separate features (e.g. a chapter in the book), merging back to the parent branch (dev) when you stabilize something.
- ***Visual programming interfaces (e.g. Blueprint) are forbidden (zero on assignment)***. The programming languages allowed are: C/C++, C# (Unity) and/or Python (Maya).
  - If you are using Unity, all front-end code must be implemented in C# (i.e. without the use of additional editors). You may implement and use your own C/C++ back-end plugin. The editor may be used strictly for UI (not the required algorithms).
  - If you are using Unreal, all code must be implemented directly in C/C++ (i.e. without Blueprint). Blueprints may be used strictly for UI (not the required algorithms).
  - If you are using Maya, all code must be implemented in Python. You may implement and use your own C/C++ back-end plugin. Editor tools may be used for UI.
  - You have been provided with a C-based framework called *animal3D* from your instructor.
  - You may find another C/C++ based framework to use. Ask before using.
- ***The 'auto' keyword and other language equivalents are forbidden (-1 per instance)***. Determine and use the proper variable type of all objects. Be explicit and understand what your data represents. Example:
  - auto someNumber = 1.0f;
    - This is a float, so the correct line should be: float someFloat = 1.0f;
  - auto someListThing = vector<int>();

        - You already know from the constructor that it is a vector of integers; name it as such: vector<int> someVecOfInts = vector<int>();
    - Pro tip: If you don't like the vector syntax, use your own typedefs. Here's one to make the previous example more convenient:
        - typedef vector<int> vecInt;
        - vecInt someVecOfInts = vecInt();
- ***The 'for-each' loop syntax is forbidden (-1 per instance)***. Replace 'for each' loops with traditional 'for' loops: the loops provided have this syntax:
    - In C++: for (<object> : <set>)...
    - In C#: foreach (<object> in <set>)...
- ***Compiler warnings are forbidden (-1 per instance)***. Your starter project has warnings treated as errors so you must fix them in order to complete a build. ***Do not disable this***. Fix any silly errors or warnings for a nice, clean build. They are generally pretty clear but if you are confused please ask for help. Also be sure to test your work and product before submitting to ensure no warnings/errors made it through. This also applies to C# projects.
- ***Every section/block of code must be commented (-1 per ambiguous section/block)***. Clearly state the intent, the 'why' behind each section/block. This is to demonstrate that you can relate what you are doing to the subject matter.
- ***Add author information to the top of each code file (-1 for each omission)***. If you have a license, include the boiler plate template (fill it in with your own info) and add a separate block with: 1) the name and purpose of the file; and 2) a list of contributors and what they did.
- ***Immediate mode is forbidden (zero on assignment)***: In this course we are studying modern graphics engineering principles; immediate mode refers to an ancient and deprecated set of OpenGL functions. The tutorials followed use the correct techniques; do not use tutorials on the internet that will lead you astray.

**Points** 10

**Submitting** a text entry box

| Due | For | Available from | Until |
|---|---|---|---|
| - | Everyone | - | - |

**GraphicsAnimation-Master-Range-x2**

| Criteria | Ratings | | | Pts |
|---|---|---|---|---|
| IMPLEMENTATION: Architecture & Design<br>Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine. | **2 to >1.0 pts**<br>**Full points**<br>Strong evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely both efficient and functional. | **1 to >0.0 pts**<br>**Half points**<br>Mild evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely either efficient or functional. | **0 pts**<br>**Zero points**<br>Weak evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely neither efficient nor functional. | 2 pts |
| IMPLEMENTATION: Content & Material<br>Practical knowledge of content relevant to the discipline and course (e.g. shaders and effects for graphics, animation algorithms and techniques, etc.). | **2 to >1.0 pts**<br>**Full points**<br>Strong evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely both efficient and functional. | **1 to >0.0 pts**<br>**Half points**<br>Mild evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely either efficient or functional. | **0 pts**<br>**Zero points**<br>Weak evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely neither efficient nor functional. | 2 pts |
| DEMONSTRATION: Presentation & Walkthrough<br>Live presentation and walkthrough of code, implementation, contributions, etc. | **2 to >1.0 pts**<br>**Full points**<br>Strong evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely both accurate and confident. | **1 to >0.0 pts**<br>**Half points**<br>Mild evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely either accurate or confident. | **0 pts**<br>**Zero points**<br>Weak evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely neither accurate nor confident. | 2 pts |
| DEMONSTRATION: Product & Output<br>Live showing and explanation of final working implementation, product and/or outputs. | **2 to >1.0 pts**<br>**Full points**<br>Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable. | **1 to >0.0 pts**<br>**Half points**<br>Mild evidence of correct and stable final product that runs as expected; end result is largely either correct or stable. | **0 pts**<br>**Zero points**<br>Weak evidence of correct and stable final product that runs as expected; end result is largely neither correct nor stable. | 2 pts |

| Criteria | Ratings | | | Pts |
|---|---|---|---|---|
| ORGANIZATION: Documentation & Management<br>Overall developer communication practices, such as thorough documentation and use of version control. | **2 to >1.0 pts**<br>**Full points**<br>Strong evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely both documented and organized. | **1 to >0.0 pts**<br>**Half points**<br>Mild evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely either documented or organized. | **0 pts**<br>**Zero points**<br>Weak evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely neither documented nor organized. | 2 pts |
| BONUSES<br>Bonus points may be awarded for extra credit contributions. | **0 pts**<br>**Points awarded**<br>If score is positive, points were awarded for extra credit contributions (see comments). | | **0 pts**<br>**Zero points** | 0 pts |
| PENALTIES<br>Penalty points may be deducted for coding standard violations. | **0 pts**<br>**Points deducted**<br>If score is negative, points were deducted for coding standard violations (see comments). | | **0 pts**<br>**Zero points** | 0 pts |
| | | | | Total Points: 10 |