Middle East Technical University    Department of Computer Engineering

# CENG 469

## Computer Graphics II

Spring 2023-2024
Assignment 1 (v1.0)
Catmull-Clark Subdivision Surface

Due date: April 5, 2024, Friday, 23:59

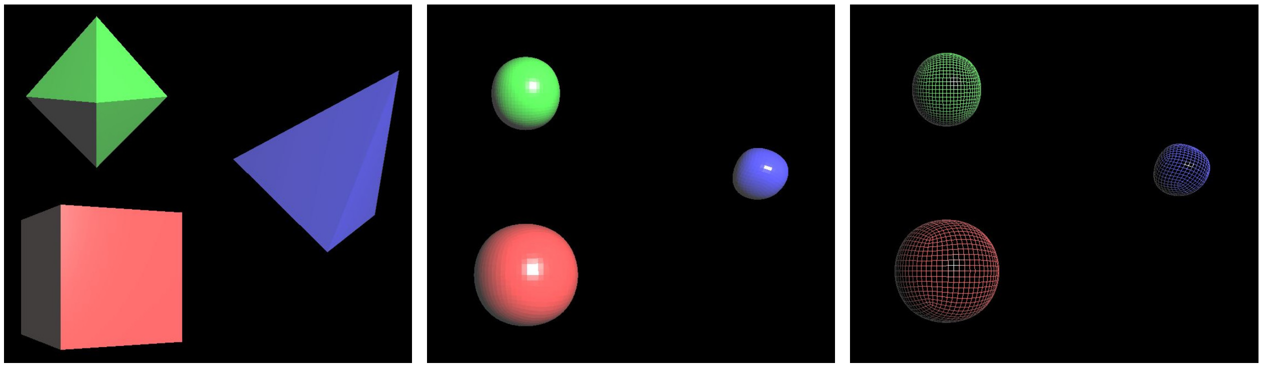


Figure 1: Output visuals after iteratively applying the algorithm four times.

## 1 Objectives

In this assignment, you are going to implement the Catmull–Clark subdivision surface algorithm and apply it to simple coarse meshes to render refined surface visuals in OpenGL. The expected end product of this assignment is an OpenGL program which:

- Runs Catmull–Clark subdivision surface algorithm iteratively on the meshes of an animated scene composition that includes at least three different moving meshes with different colors,
- Visualizes the resulting scene composition using OpenGL in three different drawing modes where the diffuse and specular shading effects are always shown on the drawings of the meshes,
- Includes user interaction capabilities with keyboard buttons to interact with the scene setup.

Some sample visuals are provided in Figure 1 and Section 4. More visuals can be found in the YouTube video we shared. The specifications are explained in Section 2. You are free to implement your own design choices as long as you obey the requirements discussed in this assignment text.

## 2 Specifications

1. A detailed description of the steps of Catmull–Clark subdivision surface algorithm is available in the following clickable resource links: *(i) Paper, (ii) Wiki Page.*

2. You can inspect, edit and use the sampleGL codebase provided to you on ODTUClass as a basis to start your homework implementation.

3. There should be at least three different meshes being animated inside the scene. The design of the scene composition is up to your imagination, but the cube mesh we shared to you needs to be available in the scene so that we see the correctness of your implementation. The scene shown in Figure 1, for instance, includes a red cube, a blue tetrahedron, and a green octahedron as the scene composition where all of them are animated. You can also design this same scene.

4. The implementation of the algorithm should work for both triangle and quad topologies, meaning faces with 3 vertices and 4 vertices, respectively. Your scene should include meshes of both topology types. Some mesh .obj files are shared to you on Odtuclass Homework page. The .obj parser code of sampleGL codebase can be edited/used to parse both topologies.

5. The *level* of the algorithm on a mesh means that the algorithm is applied that many times on the starting mesh. Level 0 means the mesh is rendered as read from its .obj file, level 1 means 1 iteration of the algorithm is applied. Initially, there should be at least one mesh in your scene setup which starts at a level that is non-zero and different then the other meshes' levels. Others might start at 0.

6. **Visualization Modes:**

   The program should have 3 visualization modes to render the scene:

   - **Solid mode:** OpenGL is instructed to draw the meshes using triangles. This is the initial mode that your program launches with. Quad faces in your code's mesh structure should be triangulated into two triangles per quad face before the draw operation. However, the visual will be seen as a quad. See the item below detailing the flat shading approach to read more details on how this visual is implemented.
   - **Line mode:** OpenGL is instructed to draw the meshes using lines. The back-sides of the meshes should not be visible, and also the meshes that are closer to the camera should occlude the meshes behind them. Two rendering passes are needed for one frame to implement this approach. In the first pass, writing to the color buffer is disabled and then all meshes are drawn using triangles as in the Solid mode (hint: glColorMask). This procedure updates the depth buffer. Then, in the second rendering pass, writing to the color buffer is enabled, and all meshes are drawn using lines. As your code should already have depth testing enabled, this second pass will only render the visible lines. The effect can be inspected in the video we shared. *glPolygonOffset* can be leveraged to prevent alised line visuals.
   - **Wire mode:** OpenGL is instructed to draw the meshes using lines. Unlike Line mode, no back-face handling is done. Therefore, all lines are visible, and no meshes block each other's visual when they overlap in the scene. Note that *glPolygonMode* of OpenGL is not changed in any of our visualization modes. It is left as its default.

7. Flat shading is to be used, meaning that each face should have its normal vector calculated, and then all vertices of that face should be set to have this same normal vector. If the face is a quad, then the two triangles that you tessellate the quad into (for Solid mode) should have their face normals calculated, and the vertices of the quad should be set to the average of those two normal vectors to have flat quad surface visual instead of a two-triangle visual. If the face is a triangle, its face normal is directly set as the normals of its vertices. Don't forget to normalize the calculated normal vectors before use. All 3 drawing modes should use this shading approach. Instead of using the pre-defined normals in the .obj files, calculate the normals as discussed above while reading the meshes from the file or after the new meshes with new algorithm levels are prepared in your code.

8. Vertex and fragment shaders are to be employed. Each mesh in your scene design should have a different diffuse color. Therefore, alongside the vertex info and the normals, the *kd* value to be used for a mesh can also be passed to the shaders as a uniform variable. You can design your shaders as you like, but the diffuse and the specular effects should be visible on the meshes while they are being animated (e.g., light should shine to the camera from a mesh when the angle is right). The video we shared can be inspected for this effect.

9. **Keyboard Buttons:**

   The user should be able to increase/decrease the level of the algorithm on all meshes of the scene during runtime as follows:

   - Initially, all meshes might start with a different level of algorithm application (though there should be at least one mesh with a different and non-zero starting level).
   - Tapping **E** button should decrease the level of algorithm application on each mesh by 1, down to the minimum value of 0. If pressed more, meshes with levels of more than 1 continue to decrease towards 0, while meshes with a level of 0 stay at 0.
   - Tapping **T** button should increase the level of algorithm application on each mesh by 1. It is expected that your code is able to render level 4 meshes for grading. More levels might cause the program to freeze while the calculations are being made.

   The user should be able to change the visualization mode of all meshes of the scene during runtime as follows:

   - Initially, all meshes of the scene are visualized in *solid* mode.
   - Tapping **M** button once should change the visualization mode to *line* mode for all meshes. Tapping it again changes the mode to *wire* mode, and tapping it again cycles back the mode to *solid* mode. The video we shared shows this approach in action.

   The user should be able to manipulate the scene during runtime as follows:

   - Initially, the meshes start their animations from their starting positions.
   - Tapping **R** button resets the whole scene to its starting configuration. The transformations and the algorithm application levels of each mesh are reset. Scene animation automatically starts again.
   - Tapping **S** button starts/pauses all the movement (animation) in the scene so that we inspect the scene during grading. When the button is pressed again, the animations

continue from where they are left off. All other buttons should still work when the scene movement is paused.

10. In your scene design, there needs to be a moment where one mesh occludes another mesh so that we pause the animation at that occlusion moment to see if your *Line* mode implementation is seen to correctly occlude the mesh behind, and also every line is seen to be visible in the *Wire* mode.

11. There is no frames-per-second (FPS) constraint in the homework, but the user interaction should feel continuous and smooth. As you increase the algorithm level more, you can see a small freeze time on that frame in your application.

12. You can use Blender software to have a ground-truth visual after the application of the algorithm. Import your .obj file to Blender, select it in the scene, click on the wrench icon on the right column to add a modifier, search for subdivision surface, click on it. This will apply the Catmull-Clark subdivision surface algorithm on the currently selected mesh in Blender. You can change its level on the right column. You can add the Wireframe modifier alongside this to inspect the new mesh in wireframe mode.

13. **Blog Post:** You should write a blog post that shows some output visuals from your work, and explains the difficulties you experienced, and design choices you made during the implementation phase. You can also write about:

    - An analysis of the algorithm level vs **FPS** (frames-per-second) of your code (maybe as a table).
    - A discussion on the moments where you got stuck, and how you have solved them.
    - Some interesting design choices you made during implementation, etc.

    The blog website you write your blog on should have the ability to automatically show the last edit date for your post which shouldn't be able to be edited by the writer.

    The deadline for the blog post is 3-days later than the original deadline for the homework. You can submit your code before the homework deadline, and finalize the blog post during the following 3 days without losing late days. You consume your late days only if you submit your "code" late. However, submitting the blog post more than 3 days after the original homework deadline will incur grade deductions.

    If you would like to put some piece of code or a link to the online repository of your homework implementation into your blog post, please do so by editing your post 3 days after the homework's code deadline, so that your code is not visible to the students who use 3 late days for the coding part. The rest of the post should already be published before the blog post deadline, which is 3 days after the original homework deadline. We will check the blog pages the moment the blog post deadline ends.

## 3 Regulations

1. **Programming Language:** C/C++ is highly recommended. You also must use gcc/g++ for the compiler if you use those programming languages. Any C++ version can be used as long as the code works on Inek machines. If you use any other programming language, be

sure that the Inek machines can compile and run your code successfully and also include a simple Readme file to describe how we should run your code on Inek machines during the grading (if you don't use C/C++).

2. **Additional Libraries:** GLM, GLEW, and GLFW are typical libraries that you will need. You should not need to use any other library. But if you still want to use some other library, please first ask about it in the ODTUClass forum of the homework.

3. **Groups:** All assignments are to be done individually.

4. **Submission:** Submissions will be done via ODTUClass. You should submit your blog post link to Blog Links forum on ODTUClass, so that the URLs are publicly available for everyone's access, using the title "HW1 Blog Post". For code submission, create a "**.zip**" file named "hw1.zip" that contains all your source code files, shader files, mesh files, and a Makefile. The executable should be named as "**main**" and should be able to be run using the following commands (any error in these steps may cause a grade deduction):

   ```
   unzip hw1.zip -d hw1
   cd hw1
   make
   ./main
   ```

5. **Late Submission:** You can submit your codes up to 3 days late. Each late day will be deduced from the total 7 credits for the homeworks of the semester. However, if you fail to submit even after 3 days, you will get 0 regardless of how many late credits you may have left. If you submit late and still get zero, you cannot claim back your late days. You must e-mail the assistant if you want your submission not to be evaluated (and therefore preserve your late day credits).

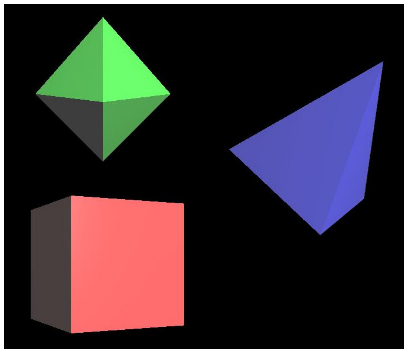
6. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get a 0 for the homework. You can discuss algorithmic choices, but sharing code between students is strictly forbidden. We also collect codes that are similar to the homework's idea from the Internet and include them in the cheating control. Please be aware that there are "very advanced tools" that detect if two codes are similar.

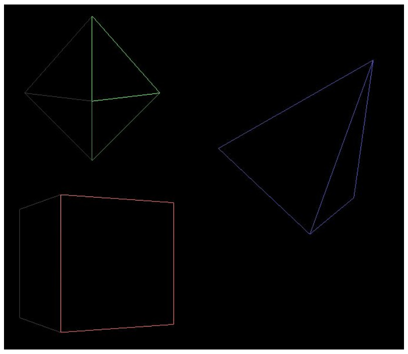
7. **Forum:** Any updates/corrections and discussions regarding the homework will be on ODTUClass. You should check it on a daily basis. You can ask your homework related questions on the forum of the homework on ODTUClass.

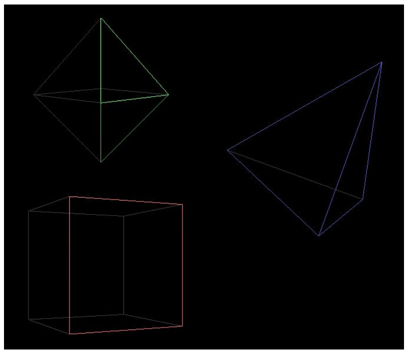
8. **Grading:** Your codes will be evaluated on Inek machines. We will not use automated grading, but evaluate your outputs visually by interacting with your program. Note that you should test your code on an Inek machine before submission, as the frame rate might not be as high as your home computer if you have a powerful PC, and might cause hangs during animations. This might require reducing/optimizing per-frame calculations in the code. Blog posts will be graded by focusing on the quality of the content.

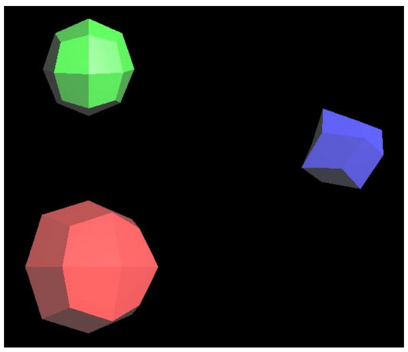
   The Supplementary section can be found in the next page.

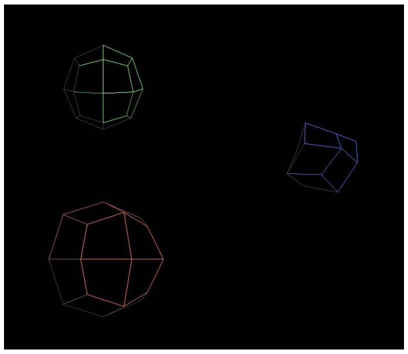
## 4 Supplementary

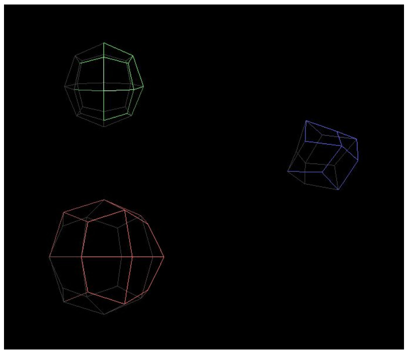

(a) Level 0 - Solid Mode

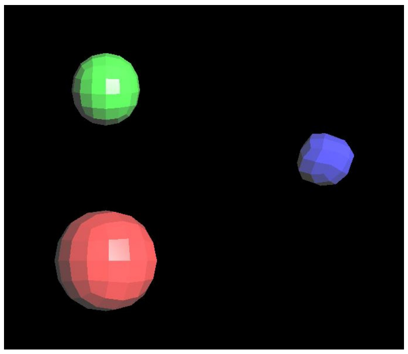

(b) Level 0 - Line Mode

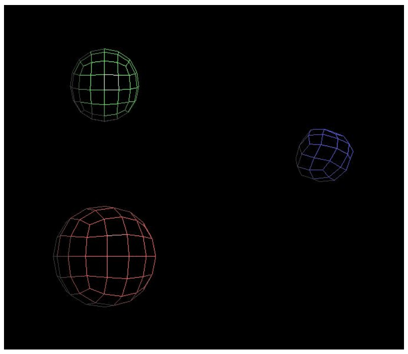

(c) Level 0 - Wire Mode

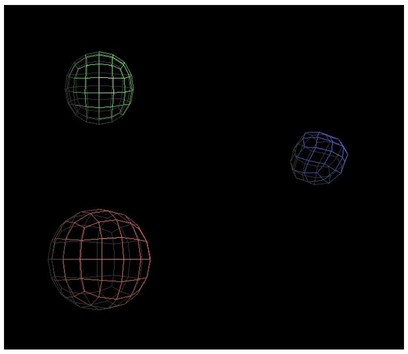

(d) Level 1 - Solid Mode

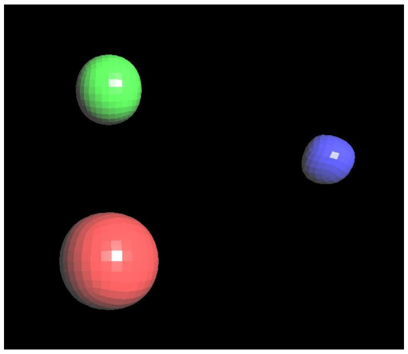

(e) Level 1 - Line Mode

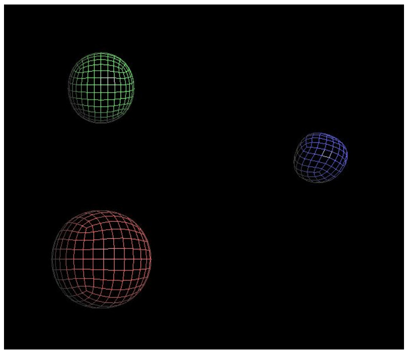

(f) Level 1 - Wire Mode

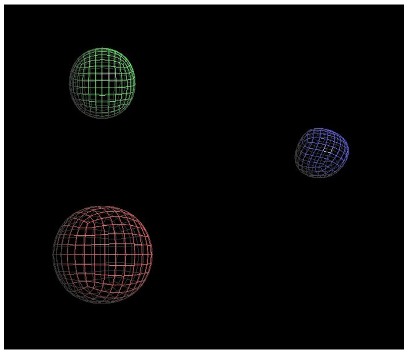

(g) Level 2 - Solid Mode


(h) Level 2 - Line Mode


(i) Level 2 - Wire Mode


(j) Level 3 - Solid Mode


(k) Level 3 - Line Mode


(l) Level 3 - Wire Mode