

CNG 477

Introduction to Computer Graphics

Homework 3 – Rendering with OpenGL

Deadline: 22.05.2026 23:59

In this assignment, you are going to implement a simple 3D game that resembles the well-known 2D Tetris game. You are going to use **OpenGL** with **GLSL** shaders.

A demo video bundled with the homework package should give you a clear idea of what is expected from your programs.

Keywords: OpenGL, GLSL, 3D Tetris, GLM, GLEW, GLFW

Specifications

1. Name your executable as tetrisGL.
2. Your executable will not take any input file. It should be run as ./tetrisGL.
3. There is a single game block, a 3×3 cube. It can be moved left and right. It falls towards the bottom at a constant speed. You must implement speed-up, slow-down, and stop options.
4. The board must be a 9×9 grid, colored differently than blocks, and should have some thickness (not a flat plane).
5. Both blocks and the board must have a grid structure with visible borders.
6. Key controls:
 - **A:** Move block left (current view)
 - **D:** Move block right (current view)
 - **S:** Speed up falling
 - **W:** Slow down falling
 - **H:** Rotate view left
 - **K:** Rotate view right
7. Although blocks move left/right in current view, in another view they move forward/backward accordingly.
8. Use **diffuse**, **ambient**, and **Blinn-Phong** shading models.
9. Use at least **one point light source** and **ambient light**. You may move the light with the camera or use multiple lights.
10. *The user must be able to select from at least two predefined materials (e.g. wood, metal, plastic, or similar). Each material must use visibly **different diffuse and specular** response under the same lighting (item 8-9), so that **Blinn-Phong** highlights and overall surface appearance clearly change. State in the report or README how to **switch** materials (e.g. keys or a simple menu).*

11. A template program is provided that draws a cube with white borders. You may start from this.
12. Gameplay must follow the demo:
 - Blocks fall **step by step**, not continuously.
 - **Camera rotation must be smooth.**
 - **No intersection** between blocks and the board.
 - Blocks cannot move outside the game area.
13. Number of steps before touching ground is up to you (e.g., 12 in demo).
14. Display **current view** (top-left) and **score** (top-right) on the screen. Show "game over" when applicable.
15. Follow standard **Tetris rules**: completed rows collapse, upper pieces fall, score updates, new blocks spawn. If a new block cannot be placed, the game ends.
16. Each time your active piece completes **one downward step** and comes to rest on the floor or on stacked cubes ("**lands**" or **locks** before the next sideways move or fall), briefly animate **only that falling piece**:
 - Apply a **short shear** along a horizontal axis (relative to **up**. for example a small skew **using the modeling matrix**) and ease it **back** to identity. You are **not** required to squash with **scaling** here; focus on shea. **Do not** change collision detection or stored cell coordinates.
 - The motion is **purely cosmetic**: which cells are filled, scoring, spawning, etc. must behave **exactly** as without the animation.
 - Keep it **brief and modest** so the shape stays readable and it doesn't **look** like cubes **cut through** the board or neighboring blocks (**watch the shear amount and pivot**, e.g. shear around the piece's **bottom** center so motion feels anchored on the stack).
17. When the game **clears one or more full rows** with the **normal Tetris rules**, the **data** for the board updates right away (empty cells, blocks above move down, score, next piece is **unchanged**).
 - You must also make it **look better** on screen: instead of every cube **jumping** to its new place in the same frame (basically teleporting), make the blocks that **drop into the gaps** move with a **short wave**, for example, **upper rows start moving a tiny bit later** than **lower rows**, so the motion feels like a ripple.
 - You can drive the timing with **shader uniforms** or with **interpolation on the CPU**; both are fine.
 - **Rules that still apply**
 - Nothing about **winning, losing, scoring, or what is legal** may change; extra motion is **only** for display.
 - When the animation finishes, what you **draw** must **match** your **grid in memory** exactly.
 - Keep the whole effect **short** (roughly **under half a second total**), so it still

feels like normal **step-by-step** play, not slower physics.

18. You can make design choices as long as gameplay matches the demo and all controls are implemented. **Step-by-step motion and smooth rotation are essential.**

Hints s Tips

1. Start early. Graphics programming takes time to get used to.
2. Write a **bounding box class** to detect collisions. Write functions to check whether a point is in a region.
3. Remember that **each frame is rendered from scratch** in forward rendering. Use data structures (e.g., `std::vector`) to store what needs to be rendered.

Bonus

You can earn **bonus points** by adding features like:

- New block shapes
- Alternative gameplay mechanics
- Visual effects

Note: Bonus is only valid if your program **fully meets the required specifications**.

Regulations

1. **Programming Language:** C/C++ (any version) for the main program; **GLSL** for shading. Provide a **Makefile**. Use GLSL version 330.
2. **External Libraries:** You may use:
 - Standard C++ STL (e.g., `std::vector`)
 - GLFW (window management)
 - GLEW (OpenGL extensions)
 - GLM (matrix operations)

The template code uses these.

3. **Sample Code:** You may modify or ignore it. It's just a starting point.
4. **Submission:**
 - Submit via **ODTUClass**.
 - Name your tar: `tetrisGL.tar.gz`
 - Include **source files** and **Makefile**
 - Executable must be `tetrisGL`

Submission will be tested with:

```
tar -xf tetrisGL.tar.gz
```

```
make
```

```
./tetrisGL
```

Any issues in these steps will result in **point deduction**.

5. **Evaluation:** Based on expected features and overall **playability** of your game.