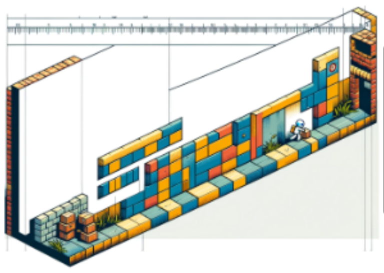

PROG2002 Graphics Programming Exam Autumn 2023

SOKOBAN

Assoc. Prof. Rafael Palomar

NTNU

Department of Computer Science

Examination paper for PROG2002 – Graphics Programming

Examination date: December, 15th 2023
Examination time (from-to): Dec, 15th 08.00am – Dec, 18th 08.00am
Permitted examination support material: A / All support material is allowed (with restrictions, see below)

Academic contact during examination: `rafael.palomar@ntnu.no`
Administrative contact during examination: `eksamen@gjovik.ntnu.no`
Phone: 61135400

Introduction

In this exam, the student will program a simplified game of Sokoban in 3D. Sokoban is a classic puzzle game in which the player needs to push boxes to their correct location in a warehouse. A playable example of the original game (in 2D) can be found here: https://www.sokobanonline.com/

The game must be developed using the following technologies:

- C++ (>= 17)
- CMake (>= 3.15)
- glad (Not GLEW!)
- glfw (>= 3.3.2)
- glm (>= 0.9.9.8)
- OpenGL (>=4.3, >=4.1 for Mac OS Users)
- Recent `stb_image`

Important information

- Read this document entirely before you start.
- Do not open Inspera in multiple tabs, or log in on multiple devices, simultaneously. This may lead to errors in saving/submitting your answer.
- The exam is an individual (not in groups), independent work. All submitted answers will be subject to plagiarism control. Read more about cheating and plagiarism here*. You are allowed to use original code you have developed during the assignments, as well as the labs code, however, any external source that has been directly added (copied) to your assignments, and therefore to the exam, needs to be properly referenced and might affect to your grade if your own contribution is not enough.
- Cite your sources in the readme file of your solution. Not citing your sources can be considered plagiarism.
- Questions are allowed provided that the purpose is ONLY clarification of the exam tasks. We won't provide any advice/direction/help to solve the tasks.

*https://innsida.ntnu.no/wiki/-/wiki/English/Cheating+on+exams

- If the need to contact the candidates during the exam should arise (e.g. in case of an error in the question set), notifications will be given via e-mail. You should therefore check your inbox from time to time during the exams.
- The deliverables need to be delivered before the deadline established for the exam. Late or missing deliverables will count as a failed exam.
- The repository allocated for your exam should be private during the course of the exam. Inform the responsible of the course if it is not.
- External repositories that are not your contribution (e.g., glfw, etc.) should be included as `git submodules`. However, external repositories that are your contribution to the exam (e.g., external framework you developed during assignments) need to be included entirely as a hard-copy.
- The solution needs to run in a 1024×1024 window.
- Faulty CMake project definitions, missing files in the project, making unable to compile/run the project, may constitute a failed exam.
- The following circumstances may lower your grade (severity may vary depending on the case):
  1. Solution not passing the CI pipeline checks (if not enabled by default in your exam repository you have to enable it yourself!).
  2. Libraries not included as `git submodules`.
  3. Including unnecessary files in the repository (i.e., generated binaries).
  4. Low performance of the solution (i.e., unjustified high number of draw-calls).
  5. Segmentation faults or memory leaks.
  6. Not cleaning properly OpenGL resources.
  7. Deviations respect to the definition of the tasks.

Withdrawing from the exam If you become ill, or wish to submit a blank test/withdraw from the exam for another reason, go to the menu (Inspera) in the top right-hand corner and click “Submit blank”. This cannot be undone, even if the test is still open.

Accessing your answer post-submission You will find your answer in Archive when the examination time has expired.

Materials allowed for the exam

- You can make use of the code your group developed during the labs/assignment.
- You can make use of any of the books and online resources listed in the course description.
- You can make use of OpenGL and relevant libraries' references such as docs.gl.
- You are **NOT** allowed to use AI assistants like Github Co-Pilot, ChatGPT or similar.
- You are allowed to use traditional code completion tools integrated in your IDE.

Deliverables

Deliverable D.1 **Code repository:** the source code of your solution should be in your `main/master` branch. You will report your repository url in Inspera.

Deliverable D.2 **Zipped source code:** a zipped copy (.zip) of your `main/master` branch source tree. You can get this from your repository at the NTNU Gitlab web-ui (there is a button for that). This will be uploaded to Inspera.

Deliverable D.3 **Issues in issue tracker:** the repository comes with a set of issues (corresponding to tasks described in this document) that need to be completed and closed. **NOTE: the completed issues need to be closed and filled up upon termination. In addition, a comment (in the issue itself) with a detailed explanation on how the issue was solved needs to be added. Not closed issues will be considered not completed; poor or no description of solution can lower the grade of the student.**

Deliverable D.4 **Video demonstration:** a video highlighting the functionality with a maximum length of 3 minutes. The video will be uploaded to Inspera.

Grading

Each individual task in this exam will be awarded a certain number of points, reflecting the degree of completion and quality, including aspects such as efficiency. The points for each task are detailed in the "Tasks" section below. It is important to note that failure to adhere to the requirements or the criteria specified in the Important Information section (see above) may result in a reduction of total points.

After the completion of all tasks, your total score will be calculated. This total score will then be used to determine your final grade, ranging from A to F. The grading scale is as follows:

F	E	D	C	B	A
0.0 - 4.9	5.0 - 5.9	6.0 - 6.9	7.0 - 7.9	8.0 - 8.9	9.0 - 10.0

Tasks

Task 1: Setting up the scenario (up to 2 points)

In this task, the scenario of the game will be generated procedurally, according to the following requirements:

Requirement R.1.1 : The main scenario of the game is a warehouse **floor** (flat surface) with **walls** surrounding it. The warehouse floor is divided in tiles, and the walls are made up by blocks. The warehouse should have a grid on the floor to indicate the division of the tiles. You may consider adding a border to the tiles or have a chessboard pattern (at this point, don't use textures to create any division effect).

Requirement R.1.2 : The warehouse should be `10x10` tiles in size, and the walls should be one tile thick (inside the warehouse floor).

Requirement R.1.3 : The player will be represented by one distinct cube (i.e., different color). The player's starting position should be randomly selected from a valid space (no wall, no box) of the warehouse.

Requirement R.1.4 : There will be 6 boxes, 6 extra pillars (wall blocks that will act as obstacles) and 6 designated locations , all randomly placed within the warehouse. Each of these elements should be clearly differentiated from the rest (i.e., a distinctive mark/color in the floor, a wireframe cube, an almost transparent cube for the designated locations...you can be creative but don't forget to indicate your decisions in the corresponding issue). Boxes, pillars, designated locations and player should not overlap on initial placement.

Requirement R.1.5: There will be a camera positioned outside the warehouse, looking towards the center of the warehouse. The camera should be able to rotate around the scenario using the keys `A` and `D`. The visualization should create depth perception. It is up to you to set up the parameters participating in the visualization, but don't use extreme values making the game difficult to play or visualize.

Requirement R.1.6: The camera should also be able to zoom in and out using the keys `S` and `W`, respectively.

Task 2: Game mechanics (up to 3 points)

In this task, the player and the game mechanics will be implemented.

Requirement R.2.1: The player must be able to move in four directions `(up, down, left, right)` using the cursor keys. At this point the movement can be tile-by-tile (e.g., discrete).

Requirement R.2.2: The player will be able to push boxes using movements with the cursor keys. Boxes can only be pushed, not pulled. The movements can be tile-by-tile (e.g., discrete). The player cannot push boxes through walls or other boxes, and cannot push two boxes at the same time.

Requirement R.2.3: Once a box is on a designated location, it will change color indicating correct placement.

Task 3: Add textures to the scene (up to 1 points)

Requirement R.3.1: Add textures to the warehouse walls and floor, and to the boxes. These textures should be colorized/blended with the underlying color, respecting Tasks 1 and 2. Texturing can be enabled/disabled by the key `T`.

Requirement R.3.2: Textures for the block/cube objects should be cubemap-based.

Task 4: Add illumination (up to 2 points)

Requirement R.4.1: Add illumination to the scene following the `Phong model`. Feel free to choose the parameters and type of illumination, but keep in mind that its effect needs to be noticeable by the evaluators.

Task 5 (up to 2 points)

For full score in any of the following tasks make use of the concept of `delta time` or elapsed time (see the Movement Speed section in `https://learnopengl.com/Getting-started/Camera`) for movements.

Choose exactly one of the following:

Task 5.A Add smooth animations

Requirement R.5.A.1: Make the player block and boxes move smoothly when they are pushed, rather than tile-by-tile; keep coherence with the game logic (e.g., boxes will change color when they are correctly positioned). You can decide about the parameters driving this animation, but make sure that it is noticeable by the evaluators.

Requirement R.5.A.2: Make an animated glow effect (cyclic increase/decrease of intensity) for the player block. It should be coherent with your illumination scheme in Task 4.

Task 5.B Day/Night cycle

Add a day/night cycle to the game where the illumination changes due to a `sun` that is visible and rotating around the scenario (keep in mind `delta time`). The sun can be represented by a point or a cube, for instance.

Requirement R.5.B.1: When the sun is above the scenario, the lighting should intensify to the maximum point (90 degrees with respect to the scenario surface). When the sun is below the scenario, the illumination should decrease, resulting in darkness. The intensity of the light should change gradually as the sun moves, creating a smooth transition between day and night. The exact values for this transition are up to you, but they should be noticeable by the evaluators. The sun should move around the scenario at a constant speed and take some time to complete a full rotation. Make it so the changes are visible to the evaluators.

Requirement R.5.B.2: The day/night cycle should follow the `Phong model` (ambient is increased/decreased and sun is a light source) and should be coherent with the illumination scheme used in Task 4.