Chapter 1

CS488/688 W15 A5: Introduction

> *"The Dominos pizza number is 745-2222. They deliver and they know where the lab is..."*
>
> Note: This quote is a bit outdated. No food or drink is allowed in the lab.

The proposal for your project is due **Wednesday, November 11th [Week 9]**. It will be marked and returned by the following class, and an updated/revised version is due **Friday, November 20th [Week 10]**. Your project submission is due **Friday, December 4st [Week 12]**, and demos will be held on **TBD [Week 13]** .

1.1 Project Specifications

1.1.1 Purpose

- Demonstrate your grasp of computer graphics.
- Implement a (set of) graphics algorithms of your own choosing.
- Use these graphics algorithms in an interesting context.

1.1.2 Statement

This assignment is the culmination of your efforts. Its primary purpose is to allow you to plan and execute a project of your own creation. So, do something interesting—write a program that does something graphical:

- Develop some graphical model...
- Create a game...
- Animate some scene...
- Explore an interaction...
- Invent a lighting model...

- Make an innovative use of the hardware features...
- Simulate the {physics,dynamics,behaviour} of something...
- or ...

1.1.3 Project Breakdown

The project is broken into three goals. The first two goals are project proposals. We will grade and comment your first proposal and return it to you. **Note that this proposal is worth about half as much to your final grade as a homework assignment.** The second goal is a resubmission of your project proposal, revised according to our comments. We will keep this version of your proposal and use it to grade your project. The third goal is the final project itself.

What we want in your project proposals is an objective list (like those found in the assignments) and supporting documentation that clearly defines these objectives. Half of the rest of your project grade will be based on this objective list. The rest of this chapter describes in detail what we expect for a project and what we want in the proposals.

1.1.4 First Goal (4 pts): Proposal Submission

You are to submit a statement of what you intend to do *in the format of a course assignment.* A hardcopy of this is to be submitted in class on the day noted in important dates.

Please Note: Your proposal will be read seriously by TAs and Instructor, and you will receive criticisms, advice, and suggestions on what you propose. These must be taken into account when submitting a Second Goal (revised) Proposal. **You will not receive any points for the Second Proposal**. The project itself is the means by which the Second Proposal is turned into credit. So it pays you to do a good job the first time around.

Special Note: You must achieve at least one objective to received any credit for your proposal. In particular, if you do not attempt a final project, you will receive 0 marks for your proposal regardless of the grade you achieved on the proposal.

A model in LaTeX format is to be found under `/u/gr/cs488/data/A5` as the file `proposal.tex`. This can be formatted using the commands

```
latex proposal.tex
dvips -f < proposal.dvi > proposal.ps
```

See `man latex` and `man dvips` for further details. There is also a `troff -ms` version in `proposal.tmpl`. Moreover, a sample of a good proposal from a past term, in PostScript form for viewing, is to be found in the file `exampleProposal.ps`

Your proposal should including the following sections:

- A **Topics/Purpose** list.
- A general, introductory **Statement** of the nature of the project. For a ray tracer, you should describe your scene, saying what features will be needed to achieve specific effects.

- A **Technical Outline** section surveying the important data structures and algorithms that will be necessary to achieve the goals, and (for ray tracing projects) lists the new commands that will need to be added to the input language.
- A **Bibliography** of a small number of papers and/or books that will be consulted, with a comment about the relevance of each to the project.
- An **Objectives** sheet containing ten different points upon which the achievement of the stated goals are to be judged. Note that if you are proposing a ray tracing project, then you should list the extra objective you implemented for A4 at the bottom of this list.

BE CAREFUL! While part of your grade will be based upon your success at reaching your goals, another part of your grade will be based upon your intelligence, understanding, comprehension, and good sense at setting goals that are neither too hard to be achieved nor too easy to be significant. Inventiveness and originality will count as well.

Grading

The TAs and instructor give points to your first proposal subjectively, by comparing the proposals against each other and against expectations. What we want to see in your proposal is that you have investigated what you plan to do for your project, and that you have decided upon a reasonable project. To receive full marks for the first proposal, you need to have

1. A clear description of your project;
2. An outline of the technical details that indicate that you understand the issues involved;
3. Appropriate technical references from reference books, texts, journals, and conference literature;
4. A list of non-trivial, pertinent, obtainable objectives.
5. Some amount of individuality in what you propose

If you are vague about your plans or objectives, if you are too ambitious or have unrealistic objectives, or if it is clear that you have not read up on your chosen subject, you will lose points.

Special note for ray tracing proposals: In your proposal, be sure to mention what your extra objective was for Assignment 4.

1.1.5 Second Goal: Revised Proposal

A revised copy and grading sheet is due after you received TA/Instructor feedback. Again, only a hardcopy is to be submitted. The objective list of this document will be used to determine part of the grade for the third goal of your project, and thus it will be retained by the instructor, and as such will contribute to your project grade. However, no points will be explicitly assigned to this goal.

By the time this is due, you should have received your graded assignment 4. If you intended to do a ray tracer project and did not get 9/10 on the assignment, you should switch to an OpenGL project. If you have to switch projects, make sure you discuss your new objective list with your instructor.

1.1.6 Third Goal (20 pts): Project Completion

You will submit physical documentation for your project. On the due date, your documentation is to be turned in to the instructor or TA. You can submit your documentation to the Computer Science office during normal working hours (8:30-12:00,1:00-4:30). **This documentation must be submitted in a bound format.** Five points will be deducted if it is not bound. Most types of binding are acceptable (three-ring binder, Cerlox, clip-on, etc.); basically, it should have a cover and be connected together (but *not* stapled or paper clipped). Projects will be returned and "medals" awarded at the final exam.

The documentation you hand in should comprise:

1. The standard material listed in the **Documentation Submission** portion of this handout, with particular attention being given to the **Manual** section, which should contain a comprehensive guide to the running of the project's program and to the formats, where appropriate, of its input, command line, interaction, output, and procedures for creating/viewing images.

   You are also required to have the usual `README` telling us how to run your program. In addition, at the end of your `README`, **you should list your objectives**.

2. An extra **Implementation** section that describes the software design considerations, including brief descriptions, where appropriate, about

   - Algorithms, data structures, and complexities,
   - Modularity, data abstraction and encapsulation,
   - Platform and system dependence or independence, global constants and configurability,
   - Input/output syntax, Lua extensions if any, pre- and post-processing,
   - Data and code sources, network and literature resources, system and local utilities, the re-use and adaptation of existing code,
   - Intercommunication, shell scripts, pipes, sockets, intermediate files, parallelism and task delegation,
   - Coding style, debugging approach and utilities, version management, testing and verifying practices,
   - Caveats, bugs, cautions, unexplored areas, assumptions, future possibilities.

3. For graduate students only *a short* report on the main points of the most important bibliographic reference(s) used for the project.

Note that the documentation you submit strongly influences your subjective mark on the project. In the past, the projects receiving the highest subjective marks had excellent documentation. Often, two projects would have approximately the same technical merit, but one would receive a much better mark due to differences in documentation. Here's a brief rundown of what we look for in documentation:

- A discussion of the interesting technical points of the project. You should note what is interesting and what is required to implement the idea.

- A map of the code. We don't want manual pages, but would like pointers to where we can find the various features you have implemented.
- References. You should note the key technical references for your project.
- Acknowledgments. You should credit any code written by other people and acknowledge people who have helped you.

To give you an example of a reasonable level of documentation, look at the sample report PostScript file to be found under `/u/gr/cs488/data/A5`. Graduate documentation is to be done in the same style, but it has to include more extensive details on implementation and a summary about any related prior work in the published literature.

Previously Written Code, Use of Other People's Code

Your project is to be code you have written expressly for the project. As such, you should not in general use code you wrote in previous terms, nor should you use other people's code in your project. However, there may be cases where you want to build on pre-existing code. If you find yourself in such a situation (whether it is code you wrote in previous terms or if it is someone else's code), you should discuss the matter with the instructor. Further, you should document in your project write-up any previously written code or code you get elsewhere. Failure to document use of such code will be considered *cheating*, even if you wrote the code yourself.

In particular, if you wrote something in previous terms that seems appropriate for your CS 488/688 project, you *may not* submit it as your CS 488/688 project. You may, however, use it as a starting point for your project; you should discuss such a matter with the instructor for the course before submitting your project proposal.

Grading

The remainder of the grade for the third goal will be based upon a subjective assessment by us, the instructor and TA(s), of how your project ranked against the others submitted this term, as well as projects like yours submitted on past terms. In total, the grade will be assigned somewhat like the judging in the Olympics for figure skating. Your list of objectives provides the scores for the "required elements," and our assessment provides the scores for the "individual merit." The individual merit judgment will be arrived at by considering the the four components of "artistic and/or innovative content," "technical depth," "software design," and "quality of documentation." This judgment will, necessarily, need to be subjective, given the wide diversity of projects. In the past we have used such criteria as:

- Artistic. Visual design and aesthetics. We will also use this category to reward humour, cleverness, originality, inventiveness, polish.
- Technical. Algorithms, mathematics, physical or optical simulation, data structures.
- Hardware techniques. Use of hardware features above and beyond those in the assignments.

- User interface. Quality of interactivity and feedback, user friendliness. Individual design of user-interface and interactive tools. Note that a user interface that is essentially ones from the assignments will net you no marks in this category.
- Code. Comments, modularity, code design.
- Documentation. Quality, thoroughness, and depth of documentation, background references, literature summaries, breadth of resources employed.

  Special Note: You will get 0 subjective marks for documentation if:

  1. your files are not where they are supposed to be, or
  2. your writeup (README + Documentation) is insufficient for us to determine how to run/use your project.
- Difficulty. This category is used to reward exceptionally difficult projects.

These criteria are an example, and they may change to suit the content mix of each term's projects. Nobody's project is expected to hit all these criteria, so we are prepared to give up to 3 points in any category. We will stop when we get to 10 subjective points, although achieving 10 points is **extremely rare**. On the average, good projects receive 4–6 subjective points, and only unusually, outstandingly, remarkably excellent projects are in the 8–10 range. The TA(s) and instructor will be as fair as possible, but standards will be high, and a "perfect 10.0" will not be given lightly—we'll probably not award it to more than one person, if that many. We are looking for polish, depth, professionalism. We are trying to find the remarkable, and locate evidence of care, thought, effort, and skill that puts the project above one that just managed to get its technical objectives.

Note that we may also give out down to -1 points in any category. This is for things you have learned during the term that you should have applied to your project, but you didn't. For example, if you do not have shading, you may get -1 in a category such as graphics techniques.

The instructor will tell you how your subjective mark was determined if you ask. However, the subjective mark is subjective; it's our opinion. If we overlooked something in our marking, then we may increase your subjective mark. However, if it is a matter of your opinion vs. our opinion, then we won't change your subjective mark. I.e., if you feel your user interface is easy to use, but we don't, then we won't change your mark. If your friends thought your project was the coolest thing they've ever seen, great! But we won't change your subjective mark.

The highest three total achievements (objective plus subjective) will be distinguished by the special awarding of a "gold," a "silver," and a "bronze" prize to be given out at the final exam. Honorable mentions may be awarded to other projects that we feel deserve special recognition.

A note about "pure" rendering (i.e., ray tracing) projects: if you look at the above list, you will see that you will easily receive points for algorithms, etc, but can not receive any for user interface issues. You should therefore pay particular attention to the following categories: Visual Design and Aesthetics (make a *very* nice image) and Documentation. In addition to written documentation, you are expected to have test images that exhibit the features you have implemented. It is insufficient to just submit your "nice image". Further, if you implement an optimisation technique, you should give timing comparisons for the renderer running both with and without your optimisations. Your documentation should state where in your code these efficiency improvements are implemented. If any background references were used, summarize these in your documentation.

1.1.7 Demonstrations

Students will demonstrate their projects on **TBD [Week 13]** ; either a sign-up sheet will be posted on the instructor's door or an online sign-up system will be used. The purpose of such a demo is for you to show that you have met your objectives, to show any extra features you have added, and to make clear what particular strong points you feel your project offers in any subjective categories. Therefore, you should base your demonstration around your objectives and around any of its possible subjective merits. Your objective points will be awarded during the demo. The subjective points are determined by the Instructor and the TA(s) as they explore your project again at a later time.

Only the TAs, the instructor, and the student giving the demo will be allowed in the lab during the demonstration. Approximately fifteen minutes will be given for each demo. You should rehearse your demonstration ahead of time, and make sure all your data files and executables are set up properly beforehand.

Be sure that your demo illustrates that your objectives have been met (you do not have to demonstrate non-graphical objectives such as optimization code).

Project Demo Requirements

- Make sure you've tested your program. Believe it or not, one person in Fall 2000 had never run his program before the demonstration. He did not get any of his objective marks.
- There should be no reason to recompile or edit things during the demo.
- For interactive projects:
  - You should be able to toggle features such as texture mapping, shadows, reflection etc in your program. Otherwise we may give you a 0 for your objective.
  - You may want to implement a cheat/god mode of your game to demonstrate some objectives. For example, if you implement particle systems that shows up when you hit 5 moving targets in a row, but it is very hard to aim, a cheat mode where you never miss will be very useful and maybe even necessary.
  - If your project was implemented at home, and some objectives did not work properly on the school machines, take screen shots of those feature and show them during the demo so you can get half marks for those objectives.
- For raytracing projects:
  - Each objective should have 1 or 2 images (for comparison) to illustrate you have completed that objective. These images should not be your final scene and should preferably demonstrate 1 objective only. Good raytracers generally have 10-20 images in addition to their final images. Rehearse showing your images. You don't want to search to find the image that shows your objective during the demo.
  - Show your best scene last.

1.2 Collected Wisdom

The following are our thoughts on various projects: what worked, what didn't work, etc., based on previous projects.

1.2.1 Finding a Project

There are several ways to find a project. First, you may have seen something in the course that you want to learn more about. This is the ideal case: just follow up on what you are interested in and see if you can make it into a project. The second way to find a project is to look for an interesting topic in a textbook, or through past issues of the conference proceedings for SIGGRAPH (older issues are published as special issues of the *Computer Graphics* journal) or *Graphics Interface*—both of which are in the library.

In both cases, you should try to find a project that is neither too hard nor too easy. The first step in deciding if a project is just right is to make up a list of objectives (Goal 1 of the project). Further, you should look at the list of subjective marks upon which we grade your project, and try to decide what subjective marks you could hope to get from your project.

As a general comment, think of the final overall effect you want from your project and come up with objectives for that effect, rather than think up objectives and build a project around it. For example, rather than say "I want to walk through a L-system forest on a fractal terrain," it'd be better to say "I want to write a golf game with trees and hills." The objectives are (roughly) the same, but the latter results in a cohesive project, while the former results in an unsatisfying project even if all the objectives are met.

If you have questions about what makes a reasonable project, ask the instructor or the TA(s). In the past, students who discussed their project with the instructor **before** submitting their first proposal would generate much better proposals and ultimately have better projects than those who did not see the instructor.

Once you find a project topic, you can use the ACM SIGGRAPH online bibliography to find papers on the topic. This bibliography on is available the World Wide Web at

http://www.siggraph.org/publications/bibliography/bibliography.html

1.2.2 Projects to be Wary of

In the past, some projects have worked better than other projects. Here's some projects that we won't accept unless you extend them in a novel way:

- Rubik's cube. This is just hierarchical modeling and transformations. Solving the puzzle is not a graphics problem.
- Fractals. While it is fine to use fractals to model something in your project, generally there isn't enough here to make a full project.
- L-systems. Same problem as fractals.
- Particle Systems. Same problem as fractals.

- Fireworks. Fireworks are a simple form of particle system and is not enough to make a project by itself.
- Solar Systems. We've had many of these in the past, and all were boring. This is a hard project to make interesting.

And here are some examples of projects that often (but not always) fare poorly when it comes to grading:

- Human/animal animation. It's hard to make creatures walk, dance, juggle, do combat, etc.
- Space games. Usually, too much emphasis is placed on the game aspects. The problem in developing good objectives is that objectives that relate to graphics are usually harder than they look. Further, these projects often get low subjective marks.
- Maze games. It's not that this is a bad project, but we've seen a lot of them so our standards are high.
- Flocking. This is harder than it looks. Further, it is hard to demo (the flock tends to fly off the screen), so if you do flocking, be sure to have special camera modes that track the flock.
- Modellers. A general purpose modeller is hard to implement. If you want to implement a modeller, design it to build something specific and expect to spend a lot of time using your modeller to man an interesting model.
- Keyframe animators. Same problem as modellers.

1.2.3 OpenGL or Ray Tracing?

One of the big decisions you need to make when deciding upon a project is whether to do an OpenGL interactive project, or a ray tracing project. It is simpler to think up a ray tracing project: You just have to make a list of extensions to the ray tracer. Interactive projects are a bit harder to devise: You need to think of an idea, determine what's interesting, and work it into a project.

Both are acceptable as projects. And while students have won awards (see below) for both types of projects, past experience indicates that interactive projects (on average) receive higher grades. Although this may in part be a reflection upon the type of students (in general) who choose ray tracing projects, there is one definite difficulty with ray tracing projects: You have to write an excellent ray tracer to get a good subjective score. And here is a hint of what we mean by excellent: **You cannot do a ray tracer project unless you got at least 9/10 for assignment 4.**

For a ray tracing project, you should think of an image/scene you want to render. Then develop your project around this scene: What features does the ray tracer need to render this scene? What features will you need to make it look realistic? Then, as you extend your A4 ray tracer, you can develop the model for your scene as you go. You should mention your scene in your proposal, and while it will be ideal if you show us a picture of the scene at your demo, it is not a requirement that you do so.

Most ray tracers will need CSG and some form of texture mapping to make a nice picture – and you can't get good subjective mark without one. However, to be a project with substance, you would need to make fairly complete/general implementations of CSG and texture mapping. And texture mapping (mapping a 2D pattern or image onto a surface) is difficult to do well, without objectionable distortion. Hence, it may be quite difficult to get good subjective marks without close attention to considerable mathematical and implementation detail. Be sure you know what the issues are before you promise to do something, particularly in rendering projects.

As discussed later, a nice scene is critical for the subjective marks of a raytracing project. We don't want to just see shapes, we want to see an actual scene. If you don't have a fine arts background, we recommend against doing scenes with just a few simple objects, i.e., you should model real life objects in a real life scene.

1.2.4 About the proposal...

The goals of your proposal are (a) to tell us what your project is and (b) convince that your project is reasonable in the sense that it's not to hard and not too easy.

Technical Outline

In this section, you need to explain the important data structures and algorithms that will be necessary to achieve your objectives. For raytracing projects, include the commands that must be added to the input language.

How well do you need to explain the data structures and algorithms? Well, you must convince us that you understand what is involved with each objective. For example, if you have bump mapping as one of your objectives, you must explain to us how you intend to map the bump map to each primitive, how the normals are perturbed, etc. If you are implementing something from a paper, it is not enough to refer to the paper, we expect you to list out the steps in the algorithm and include necessary equations. For raytracing projects, explain how your objectives will be used in your scene.

When we read your objective list, we will refer to your technical outline for details. A good technical outline will tell us exactly how you intend to achieve each objective.

Objectives

What are good, reasonable objectives? Briefly, an objective is a unit that contributes one fundamental, essential goal to your project. On the average, an objective is roughly 1/10 of your work. In a 2-3 week development time, it represents 1-2 days of planning, implementation, checking, and correction. A poorly done proposal is often characterized by attempting way too much, and that derives from not understanding clearly what is involved – either in terms of objectives or in the difficulty of each objective. You should also think about what kind of subjective marks you can get. If a project is too difficult, you may have trouble getting your objective marks.

Examples of poor objectives for extending the ray tracer follows:

1. Add spline surfaces
2. Add anti-aliasing

The first objective is too difficult and should be broken down. Both are too vague, although if adequately described in the Technical Outline then they might be okay. In this particular project, the person who wrote those objectives (no, we did not make them up) is revealing that he/she hasn't a clue about what any of those words means.

Here are good things to avoid in your objectives list:

- Code is well organised.
- Comments are good.
- Program executes correctly.

These are expected to be true as a minimum, not as objectives worth special mention. You should also avoid such objectives as

- Objects do not fly apart or distort under transformations.
- Picking works correctly.
- Program supports multiple views.

We went through these in the earlier assignments. Of course you can do this. It's not worth including as an objective. Finally, here are some objectives that are too vague:

- Interesting interaction.
- Useful feedback given on screen.
- Good use of colour.

Can't you be a bit more specific? What is the meaning of "interesting?" Who decides what is "useful" or "good?" How does the TA assess what goal has been reached?

In short: an objective should be precisely stated, clearly understood, capable of unambiguous determination about whether and to what degree it has been met. If you are using words like 'nice' or 'easy' or 'useful' or 'simple' or 'interesting' or 'realistic' in your "objective", then it's probably subjective and should be changed. Your objectives should not rehash fundamental objectives of past assignments, and they should not address basic software engineering issues that every good CS student should have mastered by now. Concentrate on objectives that are addressed specifically to the unique points of your individual project. An objective is given as a short, simple, declarative statement. For example, we can rewrite the two raytracing objectives given earlier as such:

1. Bézier surfaces is subdivided into a polygon mesh.
2. Phong shading is added.
3. Super sampling is used for anti-aliasing

One other thing to be careful of when making goals: your goals should have some technical graphics content. We commonly allow one of each of the following as objectives

- A modelling objective — this can be your final raytraced image or your OpenGL characters and scene.
- An animation objective — if you are implementing a keyframe system, creating animations with it can be one objective.
- A user interface objective — this objective can be used for some slightly complex interaction such as controlling a plane or editing a tree structure. Manipulating the camera, picking, etc, cannot count in your U.I. objective since they are already covered in your assignments. *Note: Be wary if your project needs a lot of menus and dialogs. These are very easy to implement, and so are not worth a lot of subjective points, but they can take up a lot of time better spent on achieving other objectives and fine-tuning your project.*
- An artificial intelligence objective — this objective does not really have technical graphics content, but may be useful for games. If you want an A.I. objective, it better be a good one!

Unacceptable objectives are "feature" goals without graphics content. This commonly happens with game projects. For such projects, displaying a numeric score, or giving the player extra life when a food pod is consumed may all be nice game features, but none of them are acceptable as objectives for a graphics project.

Bibliography

An excellent way to prepare for a good proposal, and to continue from there to a successful project, is to give evidence that you have informed yourself about the issues and the techniques. You do this through reading some reference material. An ideal preliminary source of ideas can be provided by looking in the index or table of contents of the texts, both required and recommended, for topics of interest. These texts will often point off to the literature for further reading. Including **specific** pertinent references to the literature in your proposals is a good way to show that you know what you are doing. For example, on 3D textures,

Bad Bibliography:

Computer Graphics, C Version, Second Edition, Hearn and Baker, Prentice Hall, 1997

(This just lists a book containing information on hundreds of topics.)

Good Bibliography:

1. **Computer Graphics, Principles and Practice, Second Edition**, Foley, vanDam, Feiner, and Hughes, Addison-Wesley, 1990, pp. 1015–1018.
2. *An Image Synthesizer*, Perlin, SIGGRAPH '85 Proceedings, pp. 287–296.
3. *Solid Texturing of Complex Surfaces*, Peachy, SIGGRAPH '85 Proceedings, pp. 279–286.

(This lists specific pages on the subject in a reference text, and it additionally lists two landmark conference articles on the subject.)

If you use reference material on the internet, you should certainly list it in your bibliography, *but a website alone is not an acceptable reference.* Websites detailing algorithms often list the source, so make sure you check those out as well.