CS606: Computer Graphics
Evaluation Booklet

Jaya Sreevalsan-Nair *
International Institute of Information Technology, Bangalore

Jan-02-2014

Important Dates

For all submissions throughout the semester, the deadline will be **11:59 pm on a designated Sunday**.

- SUBMISSIONS

checkpoint-0: 11:59 pm, Jan-12-2014;
checkpoint-1: 11:59 pm, Jan-26-2014; *checkpoint-2*: 11:59 pm, Feb-16-2014;
checkpoint-3: 11:59 pm, Mar-09-2014; *checkpoint-4*: 11:59 pm, Mar-16-2014;
checkpoint-5: 11:59 pm, Apr-06-2014; *checkpoint-6*: 11:59 pm, Apr-27-2014;

- EXAMINATIONS

Mid-term: Feb-24–Mar-01-2014; *Finals*: May-05–10-2014.
Mid-term break: Mar-02–09-2014 .

Administrivia

- **Statutory Warning:** All submissions for evaluation in this course should be original work. If not, it will be duly informed to the academic administrative authority who will take severe action on the defaulters on the grounds of academic plagiarism. **Plagiarism of any form will not be tolerated in this course.**
- **Grading:**
  - Assignments: 75% (Individual breakdown given later in the document).
  - Midterm: 15%
  - Final: 10%
- **Assignments:** There will be a total of five programming assignments and one reading assignment. The descriptions for the assignments are provided in this document. "*Start early and finish on time*" is the guiding principle for all assignments in this course. The list of papers for reading assignment will be published in Jan-2014.

*(jnair@iiitb.ac.in); shared with a co-instructor, whose details will be provided in Feb-2014

- On deadlines:
  - Assignment-n is due on *checkpoint-n*, where $n = 0, 1, \ldots, 6$.
- Rules on implementation:
  - Regarding the choice of paper for the reading assignment, no two students can work on the same paper. Hence, *first-come, first-served* policy is undertaken for the assignment of papers to the sudents. All students have to inform the instructor on the choice of the paper via e-mail. If students do not own up a paper by *checkpoint-1*, a paper will be randomly assigned to them on the day.
  - All programming assignments should be implemented in C++ using OpenGL libraries, on a Linux OS, preferrably Ubuntu or Fedora.
- Rules on submissions:
  - Several submissions can be made for the programming assignments and the technical report. Only the last submission will be considered for evaluation.
  - To incentivize early submissions and discourage late submissions the following bonus scheme will be used on the total for final grade:
    - $+1$ for submission before the designated Sunday, -0.5 for submission on the subsequent Monday or Tuesday, -1 for submission before the next Sunday, -3 any later.
  - All submissions must be sent to the instructor via e-mail.
  - Submissions should be named in the format: <RollNumber>_Assignment<Number>.* where * is tar.gz or pdf
  - Submission for a programming assignment would be a tarred-gzipped folder comprising of the source files, header files, README, subfolder containing screenshots and a Makefile.
    - There will be penalty for submissions containing intermediate files (e.g., *.o, *.C , etc.).
    - README files should contain information on sources referred to for help on the assignment, instructions on how to compile and run the application, expected input-outputs, and any notable defects/effects when running the application.
  - Submission for a technical report will be a single .pdf file.
    - Reports should be written using LaTeX and IEEE style file.
    - The technical report should follow the style of an IEEE conference paper where the body of the text should include the following sections: Abstract, Introduction, Related work, Background of the assigned paper, Discussions, and Conclusions, and References. The technical report should be original work. The report should include the findings of the assigned paper, its influence on the graphics community, and its consequences in terms of its succeeding techniques, algorithms, and/or applications.
    - A class presentation of the report will need to be done, which will also be used for evaluating the reading assignment.

- **Examinations:** There will be a written examination during the Mid-term examination week and a viva voce for finals.
  - The examinations will be based on the topics covered in the class until the date of the examination and the viva voce may also include questions based on the assignments.
  - The viva voce will be conducted by two examiners, including the primary instructor. The examination will be conducted for 15 minutes per student. Each student will be allocated a time-slot for the examination day.

- **Evaluations:** All programming assignments will be evaluated based on a demonstration and code-review. The final version of all programming assignments have to be demonstrated to the instructor before 4pm on the following Monday after the deadline. The technical report will be reviewed by two reviewers, including the primary instructor.

Assignment 0: Introductions

Grading

This assignment will not be graded.

Submission

A single .pdf file named as <RollNumber>_Assignment0.pdf, to be emailed to instructor by *checkpoint-0*.

Description

Submit a 100-200 words professional essay, not exceeding a page, in a .pdf file format. The essay should focus on the following details:

- Name, professional background (education and work experience), stream chosen at IIITB.
- A priori knowledge and practice of computer graphics, gained from a previous computer graphics course or projects.
- The expected outcome of this course in terms of knowledge and skills.

Assignment 1: GUI and 2-D Drawing

Grading

10% of final grade – 5% for demo, 5% for source code review.

Submission

A single file: <RollNumber>_Assignment1.tar.gz, containing source files, header files, README, subfolder containing screenshots, and Makefile, to be e-mailed to instructor by *checkpoint-1*.

Background

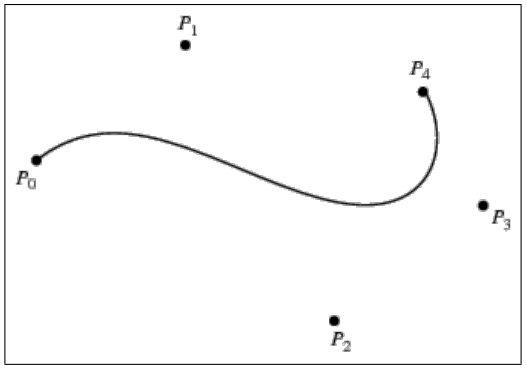

Figure 1: Fifth order Bézier curve. Image courtesy: Wolfram Mathworld.

Methods for computing Bézier curves:

1. Using Bernstein polynomial: for parameter 't' and control points $\mathbf{P_0}, \mathbf{P_1}, \ldots, \mathbf{P_n}$, $P(t) = \sum_{i=0}^{n} \mathbf{P_i} B_{i,n}(t)$, where the Bernstein polynomial $B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{(n-i)}$.
2. Using De Casteljau's algorithm which is a recursive method for computing the point at parameter 't':
   - Initial condition: $\beta_i^{(0)} = \mathbf{P_i}$
   - Recurrence relation: $\beta_i^{(j)} = \beta_i^{(j-1)} \cdot (1-t) + \beta_{i+1}^{(j-1)} \cdot t$, for $i = 0, 1, \ldots, n-j$ and $j = 1, \ldots, n$.
   - Final result: $P(t) = \beta_0^{(n)}$.

Description

Write a C++ program using OpenGL libraries, and one of the widget libraries (GLUI, GLUT, FLTK, Qt3, wxWidgets, etc.) to build an application to draw Bézier curves, as shown in Figure 1.

For the assignment:

- Read x,y,z coordinates of n control points for n^{th} order Bézier curve from a file. *Bonus points: for implementing picking the points on the screen using the mouse.*
- Display the control points using GL_POINTS.
- Using the afore-mentioned methods, compute various points on the n^{th} order Bézier curve and connect the points using GL_LINES. Implement using keys or widgets to select the algorithm.

- Implement rotate, translate, and scaling of the scene using `glRotate*`, `glTranslate*`, and `glScale*`, respectively.
- Implement adding a control point and subsequent re-drawing of the curve using keys. A control point may be added by picking a point or inputting the coordinates of the new point through command-line or widgets.

Assignment 2: 3-D Mesh Rendering with Lighting

Grading

15% of final grade – 5% for demo, 10% for source code review.

Submission

A single file: <RollNumber>_Assignment2.tar.gz, containing source files, header files, README, subfolder containing screenshots, and Makefile, to be e-mailed to instructor by *checkpoint-2*.

Description

Write a C++ program using OpenGL libraries to build a graphical application that reads in a .ply file, renders a polygonal mesh and implements lighting models on it. Use the horse data set (as shown in Figure 2 and ply tools from Large Geometric Models Archive: http://www.cc.gatech.edu/projects/large_models/.

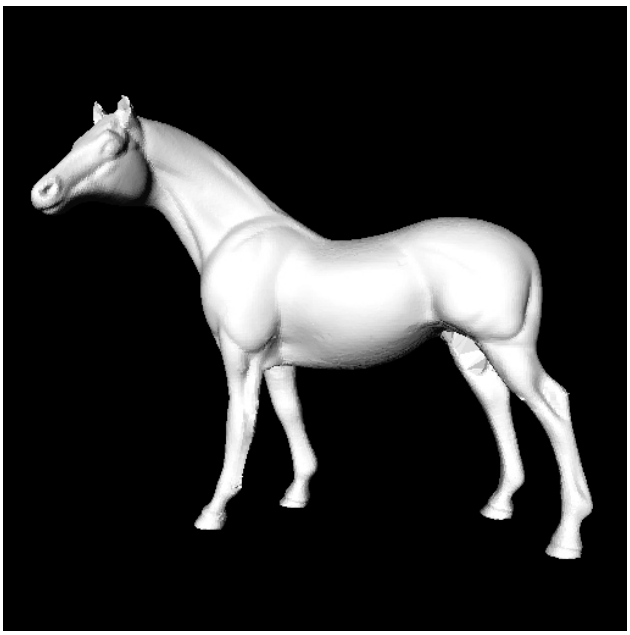

Figure 2: Horse Model consisting of 48.485 vertices and 96,966 faces. Image courtesy: Large Geometric Models Archive.

- This assignment can be implemented using one of the widget systems as used in Assignment 1.
- The application should be capable of reading any input .ply file of reasonable size.
- The application must incorporate affine transformations such as rotation, translation, and zooming. Implement rotation using quaternions. An existing library may be used for quaternions with appropriate citation. *Bonus points: for implementing quaternions from scratch.*
- Implement two lighting models: per-polygon lighting and per-vertex lighting.
- Implement the following scenarios for lighting models using one or more lights:
  1. Light moving with the object.
  2. Light remains stationary irrespective of motion of the object.

Assignment 3: Texturing & Shader Programming

Grading

10% of final grade – 5% for demo, 5% for source code review.

Submission

A single file: <RollNumber>_Assignment3.tar.gz, containing source files, header files, README, subfolder containing screenshots, and Makefile, to be e-mailed to instructor by *checkpoint-3*.

Description

Extend Assignment 2 to implement (a) Phong shading model on the horse model using shader programming using GLSL libraries, and (b) texturing on the entire mesh using an image such as Figure 3.

Figure 3: Horse skin texture. Image courtesy: Deviantart.

Assignment 4: Reading a Seminal Paper

Grading

10% of final grade – 5% for presentation, 5% for report.

Submission

A single file in .pdf format: <RollNumber>_Assignment4.pdf, prepared in IEEE conference format using LaTeX style files, to be e-mailed to instructor by *checkpoint-4*.

Description

Choose and read a paper from the list of papers, which will be published to the class in Jan-2014. These papers are seminal in the area of computer graphics. The technical report and presentation should articulate the background of the problem the paper solves, related work (if any), the methodology used, its impact on the computer graphics community, and the areas that the method affected.

Assignment 5: Interactive Surface Modeling

Grading

15% of final grade – 5% for demo, 10% for source code review.

Submission

A single file: <RollNumber>_Assignment5.tar.gz, containing source files, header files, README, subfolder containing screenshots, and Makefile, to be e-mailed to instructor by *checkpoint-5*.
Screenshots subfolder should contain the following for each of the two .ply input files:

- Control mesh and rendered subdivision surface after 3 levels of subdivision. Images showing the subdivided mesh (in wireframe) as well as the smooth shaded surface.
- Same as above but with at least 2 points moved from the original positions.

The README should contain following write up based on observations:

- Issues with implementing the subdivision algorithm. Would your implementation still work if the topology of the surface was different (e.g. if it has multiple holes). What would you need to do different if the input data was a list of triples of vertices, and not a set of indexed triangles (as in the .ply files).
- Roughly how many levels of subdivision were needed beyond which you could not perceive a significant improvement in smoothness of the rendered surface? How did this change for the different shading models?
- What were the performance issues you observed in this program?
  - How responsive was the process of editing the control mesh vertices? What shortcuts, if any, did you have to take to get good "user experience" during the editing step?
  - How would the program behave if you had a large data set? Suppose the initial mesh has 10,000 vertices and we perform > 3 levels of subdivision, then what aspects of the program do you think will need to be improved?

Description

Implement a simple interactive program that allows the editing and rendering of a mesh surface. The surface is refined through successive subdivision using a standard subdivision scheme. The user will be able to edit the shape of the surface by dragging "control points" of the surface. The behavior of the system under different levels of subdivision and different rendering schemes will be studied.

Use .ply files for the *ellell* and *dodecahedron* datasets as inputs. The datasets, as shown in Figure 4 can be accessed at: http://people.sc.fsu.edu/~jburkardt/data/ply/ply.html.

The program should have the following features:

- Read in a file with the data for the control mesh. This will be in .ply format, and typically contain a small number (tens) of vertices and edges.
- Generate the subdivision surface of this mesh using the Loop algorithm or other similar schemes.
- Display the control mesh as well as render the subdivided mesh.
  - The control mesh should be displayed so that its vertices and edges are visible. (e.g. you can use a small cube to represent each vertex. This will be useful for the pick-and-edit operation described later). The control mesh can be seen as the 0th level of subdivision of the surface.

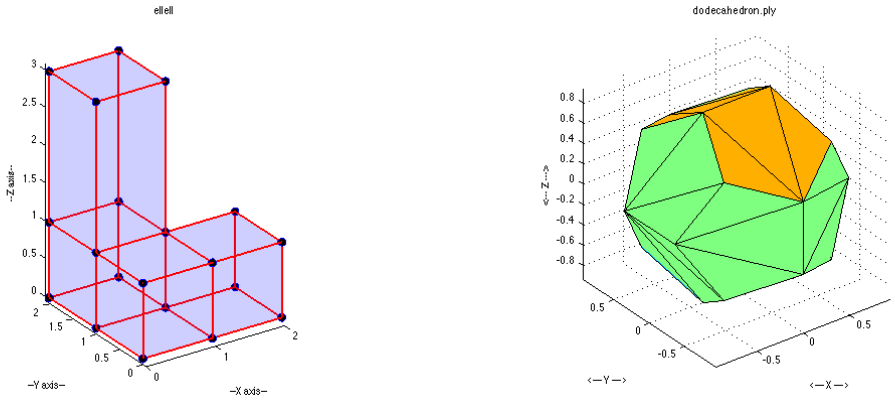

Figure 4: Ellell and dodecahedron datasets. Image courtesy: John Burkardt

  - Calculate the normal at each vertex of the subdivision surface and use this to improve the quality of shading.

- Interactively increase the level of subdivision. e.g., hitting the "+" key (or selecting a "subdivide" option in the menu) should perform one more level of subdivision. Only the final surface (at the current max level of subdivision) is to be displayed.

- Allow for the vertices of the control mesh to be interactively edited. You should be able to pick a vertex of the control mesh and "drag" it using the mouse. Assume that the movement of the vertex is in a plane orthogonal to the viewing direction. The subdivided surface should be recomputed and rendered based on the new position of the control point.

  - Render the (updated) subdivided surface as the point is being dragged.

- Provide a mechanism to change the rendering and shading style of the surface.

  - Display the edges of the subdivided mesh.
  - Render as a smooth surface using Phong shading model using GLSL.

- Common viewing capability such as zoom, pan, rotate view, position lights. Use widgets from the earlier exercises.

Assignment 6: Animation of a Hierarchical Model

Grading

15% of final grade – 5% for demo, 10% for source code review.

Submission

A single file: <RollNumber>_Assignment6.tar.gz, containing source files, header files, README, subfolder containing screenshots, subfolder containing data (optional) and Makefile, to be e-mailed to instructor by *checkpoint-6*.
Data subfolder is for data files that you used to create the models.
Screenshots should show animation at various points:

- Before the robot picks an object
- When the robot is in the process of moving the object between the belts
- After the robot has placed the object on the second belt.

Description

Implement a program to simulate the motion of an articulated robot that moves blocks from one conveyor belt to another. The objects are to be modelled as logical hierarchies. Simple forward kinematics is used to animate the robot. A simplistic view of the scene is shown in Figure 5.

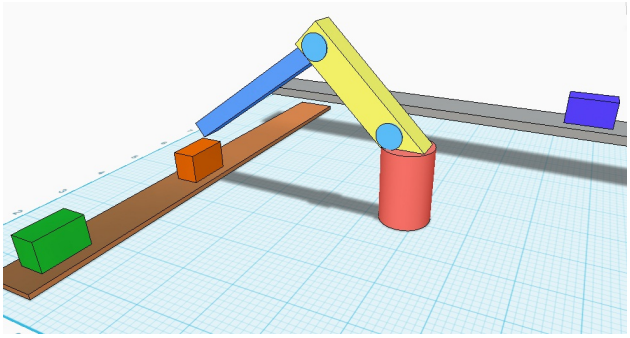

Figure 5: Objects modeled as logical hierarchies.

The program should have the following features:

- A robot consisting of at least 3 "arms": the base arm rotates about a vertical axis, the other arms rotate about a horizontal axis positioned at the end point of the previous link. Approximate shape and configuration of robot, as well as position of the belts are as in the picture above. You are free to create more elaborate robot models.
- The objects to be moved can be simple blocks, but you are encouraged to make them more interesting. (Use objects from other exercises, as an example).
- Use logical hierarchies for modeling the different parts of the scene. What sub-trees should the blocks be part of?
- Choose an appropriate speed for the belt movement, such that a block would move from one end of the belt to the other in about 5 secs. Show the objects moving along the belt, at a fixed distance apart. (You need not show the belt itself moving, though that would be great).

- Animate the robot moving objects. Robot "picks" an object from the belt when it is "close" to the tip, and then moves it to the other belt. Comes back to pick another. It should move the block from one belt to the other in about 4 secs.
  - The robot should detect when a block is close to its tip using a collision detection mechanism.
  - Compute a simple path that the tip of the robot would need to take between the belts. As an example, it could move the block up by the height difference between the belts, swing around to the second belt, and move the block horizontally to be positioned over the belt. Compute the rotations of the base and the hinges to achieve this.
- Provide for the ability to view the scene from:
  - External point and specified direction
  - From one of the objects being moved, looking towards the robot
  - From a camera mounted on an arm of the robot
- Support multiple light sources:
  - Manually positioned
  - Attached to the wrist of the robot