Pipeline TD / Software questions for Inspire Candidates

- **Answer only one question from A, B or C using either C++ or Python.**
- **Please return all source code, image sequences and notes in a single zip archive.**
- If you have time create a movie file from the image sequences to include with your submission.
- Include appropriate source code comments and cite references.
- Please use the simplest language concept which solves the problem asked.
- Don't use any third party libraries. We have provided helper classes so this shouldn't be necessary.
- C++ code should compile without warnings or errors on either Microsoft Visual Studio 2010 or gcc 4.x
- Python code should execute in Python 2.6
- Please answer the question as posed then if you have time please try extending your code in any way you find interesting.
- Please feel free to modify the supplied classes and functions.

A. Raytracing

For this question you'll investigate one of the standard algorithms used in computer graphics. Raytracing involves *casting* rays from a point through an image into a scene. When a ray intersects a surface in the scene the incident light intensity and direction is calculated and used to calculate the colour and intensity of light visible at the camera from the view direction. To calculate incident light at a surface point additional rays may be cast towards lights, or along particular directions to calculate reflection or refractions.

We have provided you with all the classes needed to extend a simple raytracer. The **Scene** class generates data for a virtual world and it is up to you to write a raytracing algorithm to visualize it. The scene consists of 3 moving spheres, a static plane, moving point light and camera. The state of the a Scene object can be queried at times from 0 to 10 seconds by calling the **Scene::SetTime (float time)** (C++) / **Scene.set_time (time)** (Python) function.

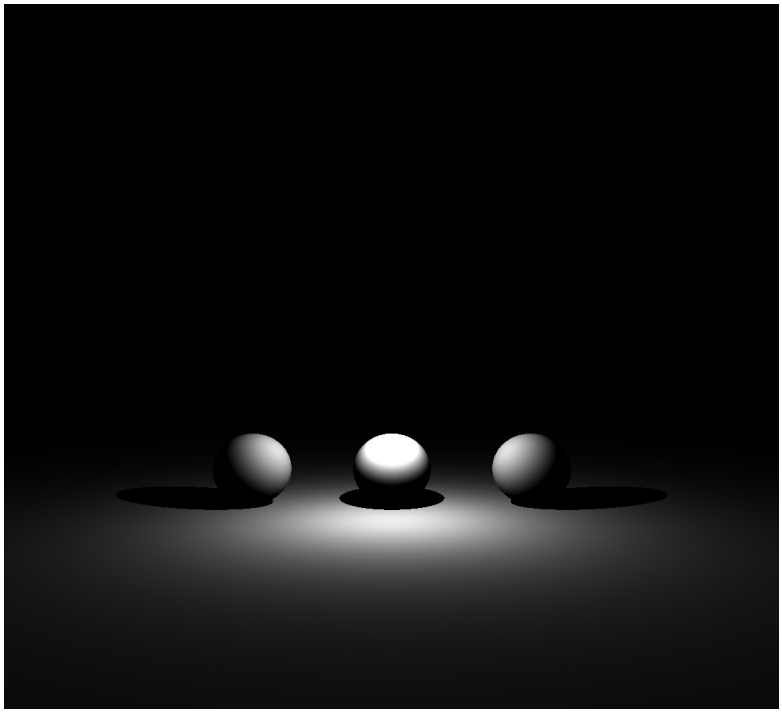

Illustration 1: Frame 8

Update either the C++ or python code to improve the realism of the sequence of images produced by the raytracer.

Suggestions for improvements:

•Make the scene more interesting

•Create a particular material (metal, glass, skin) effect

•Extended light sources or Area lights

•Glossy reflections

•Procedural textures

•Motion blur

•Depth of field

•Indirect lighting

•Remove the pixelated look in areas of high contrast

•Runtime optimizations

•Ambient occlusion

•Different geometry types

Python

Source files: Python/raytrace/raytrace.py

python raytracer.py

should produce img0001.tga img0002.tga .... img0010.tga in the current working directory

C++

Source files: C++/raytracer/raytrace.cpp / C++/raytracer/raytrace.h

Executable: raytracer

To compile: g++ raytrace.cpp -o raytracer

i.e executing raytracer should produce img001.tga img002.tga .... img0010.tga in the current working directory.

B. Image reconstruction

This problem involves creating an image from incomplete information. The function **TestData::sample** can only be called for 25% of the pixels in particular image. Using this function devise a strategy for selecting your samples and how you're going to interpolate between samples to reconstruct an approximation to the original image.

We have also provided an error function which will allow you to judge how accurate your reconstruction is.

TestData provides functions for reading from 3 sample images.

TestData::GetSize(int index) (C++) / **TestData.get_size**(i) (Python) returns the image size for the i-th image. Where i=0,1,2

TestData::Sample(int index, int x, int y) (C++) / **TestData.sample(index, x, y)** (Python) returns the color of the i-th image at (x,y)

TestData::GetNumSamples(int index) (C++) / **TestData.get_num_samples(index)** (Python) Number of times you can call sample for the i-th image.

Using the functions supplied in the **TestData** class reconstruct three images by calling **sample** the appropriate number of times for each image.

We have also supplied a function to give you an idea how accurate your image reconstruction is:

TestData::CalculateError(int index, const Image& image) (C++) / **TestData.calculate_error(index, image)** (Python)

Python

Source file: Python/reconstruction/reconstruction.py

python reconstruction.py <i> should reconstruct the ith image to a file called img<i>.tga and display the reconstruction error

C++

Source files: C++/reconstruction/reconstruction.cpp / h

Executable: reconstruction

To compile: g++ reconstruction.cpp

reconstruction <image index> should reconstruct the ith image to a file called img<i>.tga and display the reconstruction error

C. Gravity

This question in inspired by simulating stars moving under gravitational attraction. Sir Issac Newton figured out the following:

$$\vec{F}_{21} = \frac{G m_1 m_2}{|\vec{r}_2 - \vec{r}_1|^3}(\vec{r}_2 - \vec{r}_1) \quad \text{Equation (1)}$$

$$\vec{F} = m\frac{d^2\vec{r}}{dt^2} \quad \text{Equation (2)}$$

Equation 1 gives the force on body 2 from body 1.

Equation 2 defines how a mass moves when a particular force is applied.

We have supplied a source code which generates the initial conditions to simulate. The **InitialConditions** class generates 3 arrays on construction: **positions, velocities and masses.**

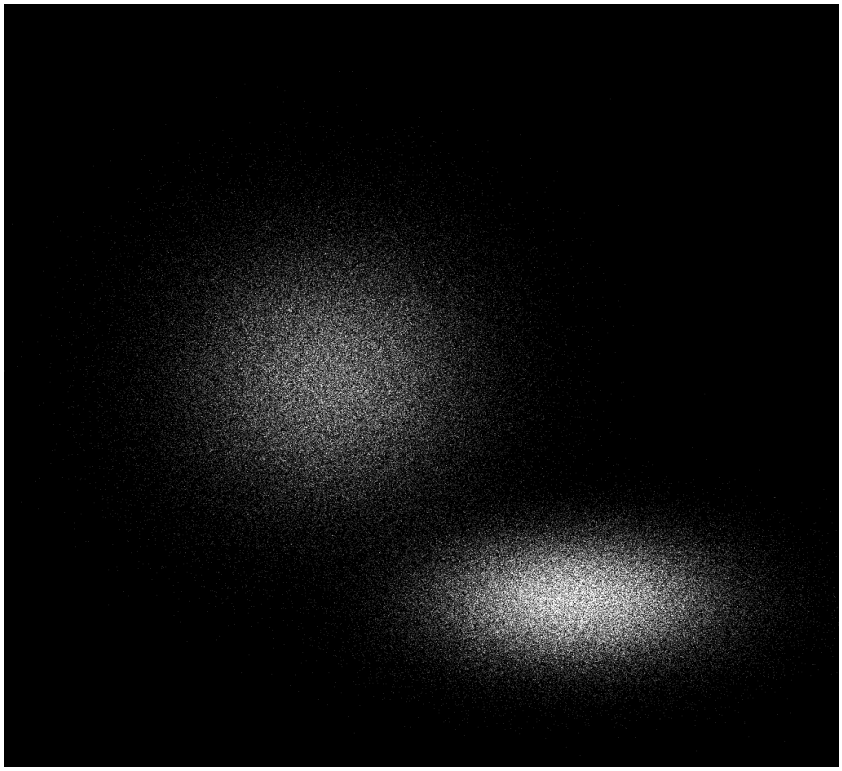

Illustration 2: Initial positions

Write code which updates the state of the system after after a given time step. For each time step you'll need to calculate the forces between all the particles of the system. Using these forces update the positions and velocities.

- Experiment with different distributions of particle positions, velocities and masses.
- Alter the force equation to generate different particle effects.

- Change the way the particles are rendered.
- How would you reduce the execution time in the language you're using?

Python

Source file: Python/gravity/gravity.py

python gravity.py <delta time> <num simulation frames> <num render frames>

should generate <num render frames> images img0001.tga, img0002.tga ...img<num render frames>.tga

Note <num simulation frames> should be multiple of <num render frames>

C++

Source files: C++/gravity/gravity.cpp / h

Executable: gravity

To compile: g++ gravity.cpp -o gravity

gravity <delta time> <num simulation frames> <num render frames> should generate <num render frames> images img0001.tga, img0002.tga ...img<num render frames>.tga

Note <num simulation frames> should be multiple of <num render frames>