Building 3D Scenes With QML

Building 3D OpenGL Scenes with Qt 5 and QML

Krzysztof Krzewniak

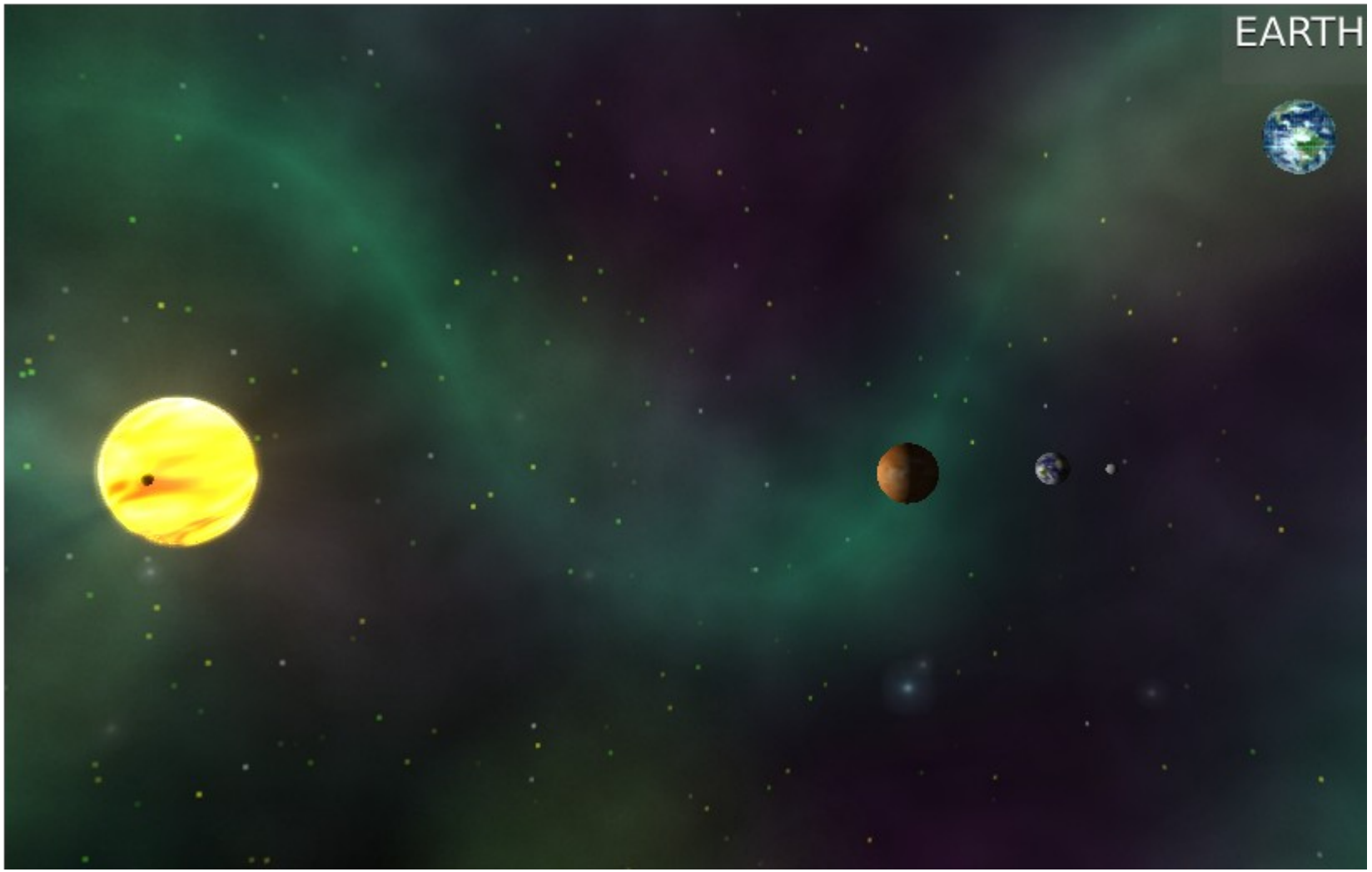
Integrated Computer Solutions (ICS)

ICS

Talk overview

- Using the QQuickWidnow's OpenGL Context to render 3D objects
- Handling the camera
- Adding scene content
- Using framebuffer objects to write filters
- Render the scene into a QQuickItem

The target for today



Hijacking the context

Hijacking the context

Be nice when hijacking

- Keep the rendering in the QSG thread
- Leave the context as you found it

Hijacking the context

Be nice when hijacking

- Keep the rendering in the QSG thread
- Leave the context as you found it

Or else ...

Hijacking the context

- Connect your rendering slot to QQuickWindow's before/after rendering signals

  Use QQuickItem::itemChange

  look for QQuickItem::ItemSceneChange

- Stop QQuickWindow from erasing your 3D scene

  Use QQuickWindow::setClearBeforeRendering

(only if rendering your contents underneath QML)

Hijacking the context

Code sample (Scene::itemChange)

Camera

- OpenGL Camera abstraction:
  - 4X4 Model View Matrix
  - 4X4 Projection Matrix
- Exposed as:
  - Camera x, y, z position
  - Camera pitch, yaw, roll
  - Projection type (Orthogonal, Perspective)
  - Field of view and clipping planes
  - Viewport width and height

Camera

Code sample (core/Camera, Camera/main.qml)

Demo (Camera)

Populating the scene

What do we need to populate the scene?

Populating the scene

What do we need to populate the scene?

- A scene item abstraction

Populating the scene

What do we need to populate the scene?

- A scene item abstraction
- A way to add items to the scene

The Scene item abstraction

Scene item properties:

- The item's x, y and z position
- Scale
- Material (keeping it simple):
  - Shader paths and custom uniforms

Scene item API:

- makeRenderPass
- cleanup

Code sample (core/SceneObject)

Adding items to the scene

Define a QQmlListProperty<SceneObject> property:

- appendSceneObject
- countSceneObjects
- sceneObjectAt
- clearSceneObjects

Code sample (core/Scene, SingleObject/main.qml)

Demo (SingleObject)

Scene filters

Increasing your scene's appeal by adding additional specialized render passes using QFrameBufferObjects

What we need:

- A render filter abstraction
- A way to add render filters to the scene
- Have the scene use render filters

Render filter abstraction 1/2

Render filter public API:

- hook – makes a filter intercept render calls
- unhook – makes a filter stop intercepting render calls
- preRender – makes a filter do its custom work
- render – makes a filter render out its results

Code sample (core/RenderFilter)

Render filter abstraction 2/2

Render filter protected API:

- createFrameBuffer - make a filter create its FBO
- bindFrameBuffer – make a filter bind its FBO
- makePreRender – make a filter do its magic
- makeRender – make a filter render its results

Code sample (filters/LightFilter, Filters/main.qml)

Demo (Filters)

Render into a QQuickItem

With all of the above in place it is very easy to have our scene or its portion rendered into a QQuickItem.

- Use a RenderFilter to redirect rendering into a FBO
- Use a QQuickItem and QSGSimpleTextureNode to render into QML

Code sample (core/textureoutputfilter)

Done

Code sample (LightDemo/main.qml)

Demo (LightDemo)