PGD Community Engine

Project Manifesto

# Introduction

The purpose of this document is to serve as a manifesto for the Pascal Game Development ([http://www.pascalgamedevelopment.com](http://www.pascalgamedevelopment.com/)) Community Engine project and to bring together the design suggestions that have been put forward following the initial request for design documents.

# Project Goals

It is perhaps worth covering the goals of the project at this point. There have been a number of graphics engines produced for the various Pascal compilers over the years. However, these are normally the work of individuals and in many cases have limited their targeting and focus to that required by the original author.

Whilst many of them are great, they often lack good quality documentation suitable for new users and new features/platform support can take a while. They can also suffer when the enthusiasm of the author wains, with users left hanging waiting for updates and bug fixes.

This project is aiming to address many of these issues by bringing a skilled community together to work on a single engine. With many engineers working on the project it is hoped that we can provide quick turn around on bug fixes etc. but also provide good quality documentation and support that could help encourage the use of Pascal as a language for making games.

At the time of writing, the project is hoping to target Embarcadero’s Delphi (we are aiming to support version 7 onwards – TBC) and the excellent Free Pascal compiler. The target operating systems are:- Windows (Delphi/FP), Linux (FP), Mac OSX (Delphi/FP), iOS (Delphi) and Android (Delphi/FP). *Project team – Please, if you believe this list is incomplete or inaccurate, please let me know ASAP. Athena.*

# Project Organisation

At the time of writing, the project does not have any real formal structure in terms of responsibility and hierarchy. This is intentional. Having a structure and hierarchy can place undue pressure on individuals if they are the only person in the project who is ‘allowed’ to perform certain tasks. A formal structure can also preclude people from indulging their own passions if that ‘position’ is already filled.

As the community manager and initiator of the project, overall control of the facilities (the version control repository, chat channels etc.) resides with Christina Louise Warne (aka AthenaOfDelphi). “I am going to do my utmost to ensure this project is a success and provides a fantastic resource to the Pascal Game Development community. To this ends, I may find myself in a situation where I have to make a decision that goes against popular opinion, but which is in the interests of the wider project and community. I’m sorry if this causes anyone a problem, but someone has to take overall responsibility for the project and ensure the wider goals are met.” CLW.

# Proposed Architecture Overview

Following a request for design documents and a discussion between those PGD members who signed up as developers, a number of documents were submitted. These appear to be suggesting a similar architecture which is outlined below.

The architecture being proposed provides a modular approach that separates the end user of the engine from the complexities of the cross platform capabilities and provides a means by which only necessary functionality is attached to game entities thus minimising unnecessary processing overheads.

It is the intention of the project to target both OpenGL and DirectX (with the early focus being on OpenGL simply because it is supported on all platforms), and to a large extent, separate the user from the differences between these two APIs to allow the user to focus on their game design rather than the technical complexities of the graphical API in use.

The focus will be on supporting 3D, with 2D support being provided using 3D methods. This will allow a greater range of effects and features to be utilised to the full. Where appropriate both high and low level interfaces will be provided allowing more experienced users a greater degree of control.

# Proposed Design

## Sub-systems

### Application

Provides platform specific interfacing for application initialization and interfacing. Other sub-systems (such as Input) will use these interfaces to manage their interaction with the platform. The application sub-system also provides internal and user defined message handling along with configuration handling (e.g. user configured key mappings).

### EntityManager

Provides management for all game entities (e.g. items, visible items, resources, materials, cameras etc.). The entities form a hierarchy, with each being fully described by a set of published properties specific to it’s type. Entities can be linked using type safe links that use absolute or relative hierarchy paths, with detaching/destroying of linked objects being handled automatically as required.

### Scene Graph Database

Stores sub-sets of the entity hierarchy (forming scenes) and allows queries to be executed against the scenes. The database should maintain optimized information about the scenes relating to certain entity types only (e.g. positionable, visible, updateable and audible). Some example queries may be:- findVisibleThroughCamera(camera), findInRange(position,range) and findVisibleInRange(position,range).

### Renderer

Provides the Graphics API (GAPI) interface. his sub-system manages GAPI state, buffers, textures etc. and is responsible for handling all drawing within the engine.

### Core

Provides the main extension point for user applications. The core should implement the main loop and orchestrate the interaction between the various sub-systems based largely on information contained in the scene graph database.

### Input

Provides device specific input handling. The input sub-system converts platform independent engine messages to input events and provides a specific controller input, using the application sub-system to communicate with the platform and hardware.

### Audio

Provides the Sound API (SAPI) interface. This sub-system manages SAPI state, audio buffers, streaming etc. and is responsible for handling all audio within the engine.

### Physics

Provides the Physics API (PAPI) interface. Manages the exchange of motion related data between the physics engine and the PGDCE entities.

### 2D

Provides the 2D drawing interface for the engine allowing sprites and overlays to be rendered using the selected renderer.

### GUI

Provides the graphical user interface (GUI) functions user can use to create forms etc. for use in their application. This will require a collection of classes to be created to represent the required form layouts.

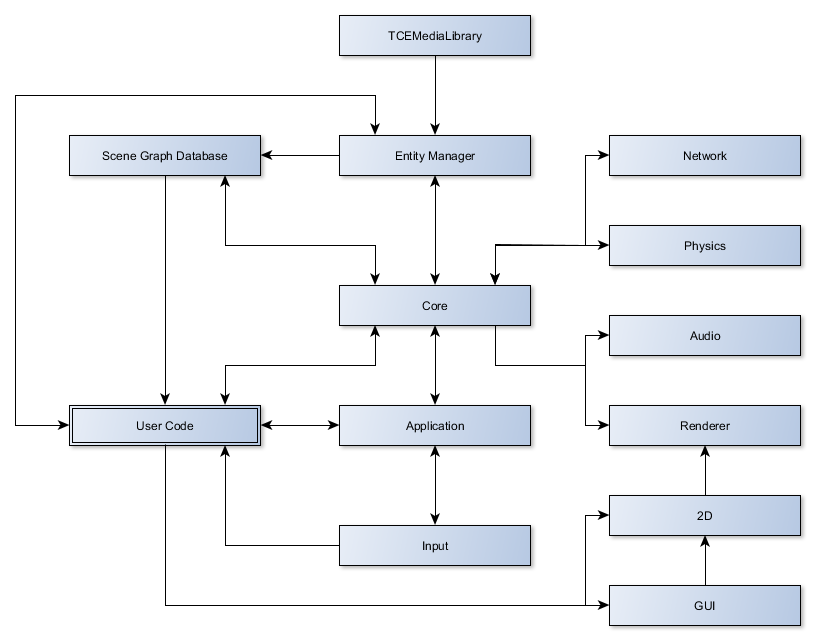
### Network

Provides networking API (NAPI) interface. Manages the exchange of data with other PGDCE instances or server software to implement things such as on-line high score tables and multiplayer games.

### MediaLibrary

Provides a random access media storage facility for game related media (e.g. sounds, images, models etc.). *Project team – Seems that this overlaps with. Athena.*

## Block Diagram



## Entity System

A key component of the engine will be the entity system. All entities will be based on TCEBaseEntity which will provide base functionality such as management of children and hierarchy traversal routines such as nextSibling etc. Descendant entities will extend the base adding the properties required to fully describe the specific type. For example, TCEPositionableEntity would add a position vector and orientation quaternion. *Can anyone from the project team recommend good reading material for 3D maths and such like? It may be nice to build a list of recommended reading so we’re all talking the same language… a glossary may also be a good plan to avoid any ambiguities - Athena*

As mentioned in the TCEEntityManager description, entities can be linked using hierarchy paths. Such links can be relative or absolute and provide a type agnostic means of building the hierarchy.

Entities can be extended by adding components to them. The components are re-usable allowing a single component to be used by multiple entities (where appropriate localised data storage is also added to the entity – for example, to ensure that multiple instances of an animated item are not all perfectly synchronised).

The initial design includes the following components:-

* TCEUpdater – Initiates periodic calls to TCEBaseEntity.update(tick) method
* TCEMesh – Encapsulates geometry information and turns a game entity into a visible entity
* TCESound – Binds a sound source to an entity
* TCECollisionVolume – Provides a collection volume and allows an entity to take part in collision checks
* TCEMaterial – Defines a material for a mesh (shaders, textures, blending etc.)
* TCELight – Binds a light source to an entity

To optimize rendering and other processing (particularly of indoor scenes) the world defined by the entity hierarchy can be sub-divided into zones (TCEZone). A zone consists of a closed set of visible entities (a room for example). Other entities (beyond the bounds of the zone) are only visible from inside the zone through open portals which can represent features such as windows and doors.

A portal (TCEPortal) itself is a convex polygon that connects two zones. The portal can be inactive (closed) or active (open), graphical representation of it’s state can be achieved by changing the mesh or material attached to it.

### Entity Usage Examples

Please note these are illustrative only and do not constitute a statement that the final classes/implementation will operate in this manner, nor should these examples be considered as examples of the coding style to be used in the project.

#### Rendering a 3D model

Var

Mesh : TCEMesh;

Material : TCEMaterial;

Model : TCEGameEntity;

…

Mesh := TCEMesh.LoadFromFile(‘…’); // Load the vertex data

EntityManager.Root.Attach(Mesh);

Material := TCEMaterial.Create();

Material.setShader(stFragment,’…’); // Setup material properties

EntityManager.Root.Attach(Material);

Model := TCEGameEntity.Create([Mesh,Material]);

EntityManager.Root.Attach(Model);

#### Loading combined material/mesh

Var

ObjectTree : TCEEntity;

…

ObjectTree := CEMeshUtil.LoadFromFile(‘…’); // Load hierarchy, return root

EntityManager.Root.Attach(ObjectTree);

#### Rendering a number of the same procedural mesHes (e.g. trees)

Var

Mesh : TCETreeMesh;

Tree : TCEGameEntity;

Material : TCEMaterial;

…

Mesh := TCETreeMesh.Create();

EntityManager.Root.Attach(Mesh);

Material := EntityManager.LoadNode(‘material.cen’);

EntityManager.Root.Attach(Material);

For I := 1 to 1000 do

Begin

Tree := TCEGameEntity.Create([Mesh]);

EntityManager.Root.Attach(Tree);

Tree.SetPosition(CEVec3(Random(100), 0, Random(100)));

End;

## Material System

At the low level, a material will be defined as a hierarchy of techniques and passes. For example:-

* Material 1
  + Technique 1
    - Pass 1
  + Technique 2
    - Pass 1
    - Pass 2
* Material 2
* …

During the initialization of the material, techniques that are not applicable (i.e. unsupported by the GAPI) will be filtered out. The renderer can then choose from the remaining techniques based on distance and other parameters that affect the desired level of detail (LOD).

Based on this flexible low level capability, various high level systems should be possible. For example:- Creation of shaders based on the type of surface and the light sources or the creation of materials in a visual editor.

# Project Developers

If you would like to sign up as a developer on the project, there are forums dedicated to the project at Pascal Game Development ([http://www.pascalgamedevelopment.com](http://www.pascalgamedevelopment.com/)). Head on over, register an account with the site and signup as a developer.

You will need a Github account as this is where the project’s version control repository is being hosted (<https://github.com/PascalGameDevelopment/PGDCommunityEngine>) and you will need to abide by the coding standards for the project (available in the repository). If you want to join in developer meetings you’ll need an IRC client to join #pgdce on the Freenode IRC network.

But what if you’re just starting out?

The project is open to anyone who is willing to help. Help can come in many forms including testing and documentation. If you’re just starting out and want to learn, signup and start out by taking on some of the smaller less technically complex tasks and then as you learn and gain confidence take on more challenging items.

The one thing we ask is that you respect all other project members and the members of any sites we may use in relation to the project (Github for example). Abuse of any kind will not be tolerated!

# Choice of Licence

The choice of which licence to use for the distribution of the project has yet to be made. Key considerations are:-

* Ability to use the engine for commercial and non-commercial endeavours without having to release the project specific code under an open source licence – This pretty much rules out all flavours of the General Public Licence
* We would like that the use of the engine is properly attributed to the engine project and the authors of said project, either by way of a licence file that is distributed or possibly a splash screen

# Pascal Game Development

Pascal Game Development ([http://www.pascalgamedevelopment.com](http://www.pascalgamedevelopment.com/)) is an on-line community of like minded individuals who choose to use Pascal (mainly Delphi and Free Pascal) to make games. The community and the Pascal Game Development Community Engine (PGDCE) project are currently being managed by Christina Louise Warne (aka AthenaOfDelphi). If you would like more information about the community or the project please contact Christina on ***athena at pascalgamedevelopment dot com***.