PROCEDURAL RUNTIME 2.4 WHITEPAPER

Abstract

ArcGIS CityEngine is based on the procedural runtime, which is the underlying engine that supports also two GP tools in ArcGIS 10.X and drives procedural symbology in ArcGIS Pro. The CityEngine SDK enables you as a 3rd party developer to integrate the procedural runtime in your own client applications (such as DCC or GIS applications) taking full advantage of the procedural core without running CityEngine or ArcGIS. CityEngine is then needed only to author the procedural modeling rules. Moreover, using the CityEngine SDK, you can extend CityEngine with additional import and export formats. This document gives an overview of the procedural runtime architecture and capabilities.

Esri R&D Center Zurich, Förrlibuckstr. 110, 8005 Zurich, Switzerland

Introduction

With the addition of CityEngine to Esri's family of desktop applications, the need for a tighter integration of the procedural 3D technology with Esri's line of 2D and 3D offerings emerged. The first step with CityEngine 2012 was extending the support of GIS data formats for seamless interoperability. With CityEngine 2013, the procedural core technology was modularized and now all Esri desktop applications use the same engine for procedural 3D model generation. Additionally, customers from the 3D content creation and entertainment industry asked for procedural technology independent of CityEngine for their applications and in-house pipelines which we wanted to answer with an independent procedural engine. Figure 1 gives an overview of the software eco-system in which the procedural runtime lives.

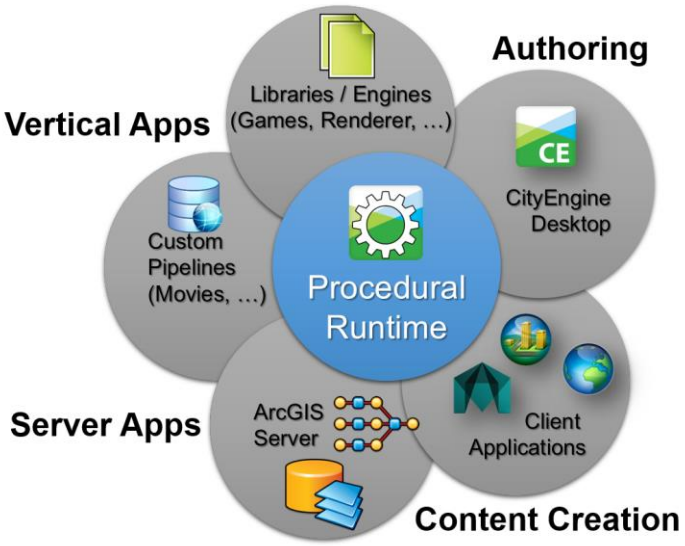

Figure 1 - The procedural runtime eco-system

These are the main driving forces behind the development of the ArcGIS procedural runtime. The runtime is a software module that essentially encapsulates the procedural core of CityEngine into a few shared libraries that can be used independent of CityEngine by any client application running on Windows, Linux, or Mac OS. In addition to that, the runtime has been redesigned to better fit large-scale multi-core and server deployments. The goal of the CityEngine SDK is to offer a powerful yet convenient toolkit for procedural modeling needs.

Overview and Architecture

Generally speaking, the procedural runtime transforms features (2D or 3D meshes) and attributes into 3D models using procedural rules. Figure 2 highlights this process.

Figure 2 - The procedural generation process. Features (2D or 3D meshes) together with attributes and rules are used to procedurally create 3D models.

Figure 3 gives an overview of the runtime's software architecture. A client application (e.g. CityEngine [1]) invokes the runtime through the PRT API [2] whose key functionality is to trigger the procedural generation of 3D geometry which is performed by Shape Processing Units (SPUs [4]). In addition to the PRT API invocations, 3rd party software can hook into the generation process by extending the runtime through decoders, encoders and adaptors using the PRTX interfaces [3]. External resource access (e.g. for 3D models or textures) and caching is handled by the data backend [5]. The callback interface [6] closes the loop and communicates the generated 3D models back to the client application.

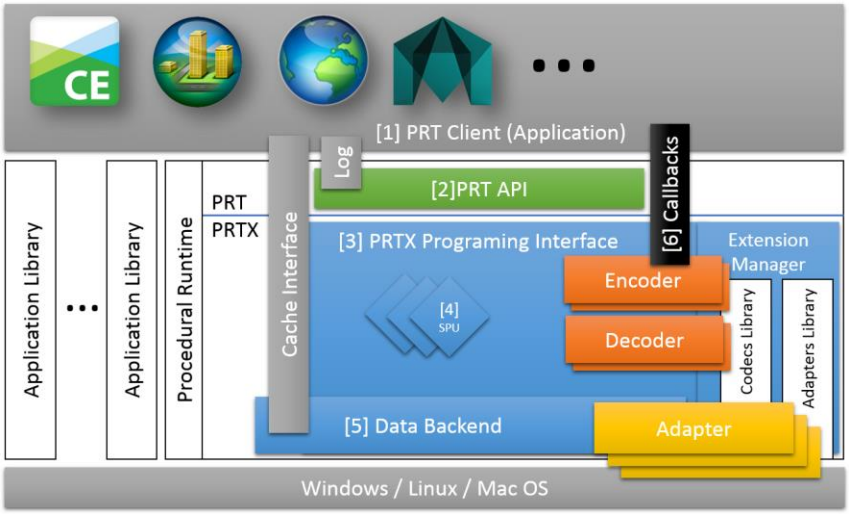

Figure 3 - Procedural runtime overview

The two worlds: PRT and PRTX

The procedural runtime consists of two worlds: PRT and PRTX. PRT is the API for triggering the generation of procedural 3D content. PRTX is the extension interface of the procedural runtime and is a collection of classes and implementation guidelines that allow extending it by custom codecs (e.g. for 3D or raster file formats) as well as for accessing non-file based data sources (e.g. an asset library in a database).

PRT – Procedural Runtime Client API

PRT is a relatively concise API which allows you to submit generation jobs to the runtime and query the runtime for its capabilities in terms of encoders, decoders, and rule files. The API minimizes the use of C++ constructs for compiler and C-runtime compatibility. Compilers sharing the same Application Binary Interface (ABI) should be able to compile and link against the provided libraries and headers without special configuration.

PRTX – Procedural Runtime Extension Interface

The PRTX programming interface is a larger collection of classes and functions that allow fine grained access to the data structures created during procedural 3D geometry generation. In contrast to the PRT API, the PRTX programming interface uses all available C++14 features. PRTX extensions are assumed to be provided in their own shared libraries and have to follow the same rules for compilation and linkage as the procedural runtime itself.

The following table highlights a few common use cases and their PRT and PRTX implications:

Use case examples	**PRT**	**PRTX**
Procedural 3D model generation in custom client application with the supplied codecs and callbacks (e.g. command line tool)	- Invoke PRT API from client application - Use one of the supplied callback implementations for results	
Integrate procedural 3D model generation with existing Digital Content Creation (DCC) application	- Invoke PRT API from DCC application - Implement custom callback for transferring 3D geometry data back to DCC application	- Write encoder for in-memory 3D geometry data suitable for custom callback implementation (or custom output format)
Create a language binding for PRT	- Wrap PRT API calls for target language - Implement custom callback for transferring 3D geometry data back to the caller	Write encoder for in-memory 3D geometry data suitable for custom callback implementation (or custom output format)
Extend CityEngine with custom export functionality		Write encoder for custom output format
Extend CityEngine with custom asset reader functionality		Write decoder for custom input format
Add custom asset repository access	Use PRT API with URI scheme for custom asset repository	Write adaptor for custom asset repository URI scheme
Custom (persistent) cache implementation / caching policy	Implement the PRT cache protocol	

Design Principles

The design principles outlined below are guiding the architecture as well as the API and interface design of the procedural runtime. The goal is to ease the integration of in a broad range of application scenarios such as 3rd party desktop application extensions (plugins), hosted 3D services, or entertainment pipelines.

CGA Rules and Assets

CGA (Computer Generated Architecture) is a property, domain specific procedural modelling language which uses CGA rule files to define rules (shape transformations) by means of CGA operations. CGA files are compiled by the CGA compiler into a binary representation called CGB (CGA-Binary) which are consumed by the procedural runtime. The CGA compiler is built into the CityEngine desktop application.

CGA defines operations (e.g. `i()` or `texture()`) that reference external assets such as 3D models (e.g. a Collada file) or textures (e.g. a TIFF file) by *symbolic names*. Thus for most CGB files, supplementary assets are required in order to generate the expected 3D result. In order to translate a symbolic name to a resource access, the procedural runtime uses a data structure provided by the client application to translate symbolic names to URIs used for resource access.

A *rule package* is an archive file with an `rpk` extension that encapsulate a CGB file and its supplementary assets. Rule packages contain a CGB, 3D assets, and textures. Rule packages are usually created with CityEngine but their format is documented and 3rd party applications can create their own RPKs if required.

Modularity

Since the procedural runtime targets a broad range of use cases, it has a modular and extensible design enabling fine-tuning for a specific environment. Currently the procedural runtime can be customized with the following 3rd party implementations:

- Callback interface: Output handling is entirely delegated to the PRT callback interface.
- Logging: Clients can add their own log handlers to the runtime logging facility.
- Adaptors can register themselves for a specific URI scheme and are responsible for resolving an URI. Several adaptors are supplied and 3rd party developers can provide their own implementations.
- Codecs translate between an external representation (e.g. a 3D file) and an internal data structure (e.g. a Geometry). There are two kinds of codec: decoders which read assets from external representations (e.g. the JPGDecoder or the OBJDecoder) and encoders which transform internal representations such as a geometry to an external representation (e.g. the FBXEncoder). Several codecs for common raster and 3D formats are supplied and 3rd party developers can provide their own implementations.
- Caching: Persistent caches or specific cache policies can be implemented through the PRT cache protocol.

Memory Management

- All PRT objects which are created through the API inherit from a common base class and must be disposed by calling a dedicated member function. This ensures that object are allocated and freed by the same C-runtime and make the client implementation independent of the PRT C-runtime version.
- PRTX utilizes shared pointers whenever possible. Therefore procedural runtime extensions generally do not need to care about freeing memory and pointer ownership.

Data Types

- In order to minimize locks, most PRT objects are immutable.
- The procedural runtime is stateless. The client can freely destroy PRT objects after calls to the API.
- The PRT API and PRTX interfaces use only double precision floating point numbers.
- Although the API is unit-less, the procedural runtime implicitly assumes that values and coordinates are in meters and Cartesian where appropriate.

- PRT API and PRTX use wide strings to avoid character encoding ambiguities (essentially using UTF-16 with the Basic Multilingual Plane) whenever possible.
- PRT objects can be converted to an XML representation which simplifies and homogenizes debugging, logging, and serialization.

Multi-Threading

The procedural runtime is designed and implemented with today's multicore processors in mind. It supports coarse-grained parallelism through its thread-safe API. The implementation minimizes global locks, and parallel invocations of the PRT API typically progress concurrently. The SPU implementation may use fine-grained parallelism when appropriate.

Despite the procedural runtime's multi-threaded implementation, PRT API clients as well as PRTX codecs and adaptors can ignore concurrency most of the time because of the runtime life-cycle management for these instances.

Error Handling

Because of the limited C++ usage for the PRT API, all PRT API calls return a status value to indicate success or abnormal conditions and never throw exceptions. However, PRTX uses C++ exceptions extensively and 3rd party code can throw exceptions if appropriate.

Internationalization

While great attention has been paid for supporting a wide range of localized environments (UTF-16, URI percent encoding, use of XML) the procedural runtime itself is English only. Since most of the runtime data has symbolic names anyway, mapping them to localized user interface strings should be straight forward in a client application.

Conclusion

This document gives just a brief overview of the ArcGIS procedural runtime. Please check the SDK documentation and examples for a detailed descriptions of classes and member functions as well as the CGA help for more information on Computer Generated Architecture.

The goal of the CityEngine SDK is to offer 3rd party developers a powerful procedural modeling toolkit in the most convenient way. Do not hesitate to contact Esri if you have specific needs, comments, or feedback regarding the procedural runtime and we will evolve the product accordingly.