Porphyrograph: New features for summer 2024

- a full scenario *PG_full_scenario.csv* contains all the possible variables used for controlling PG. Any scenario can only use a subset of these variables. Additional variables will not be taken into consideration unless they are added to the full scenario, and PG is recompiled after running the script *PG_cpp_source_generator*.

- the former header file data have been dispatched as follows:

    - the project data (UDP client/server, cameras, shaders) have been added to the scenario after the scenes and before the previously existing data

    - the project variables such as screen position have been added to the full scenario

    - the structure variables such as screen size have been located into a new include header file *pg-header.h* together with other constants from *pg-all_include.h*

- the *PG_source_generator.py* has been split into two scripts:

    - *PG_cpp_source_generator* generates C++ code to bind the C++ variables with glsl variables in the shaders (used for recompiling PG after adding new variables in the full scenario)

```
# code generated from full scenario for binding C++ and GLSL variables
include/pg_script_header_Core.h (externals for full scenario-based shader variables)
pg_render_body_Core.cpp (loads C++ variable values into tables passed to shaders as uniforms)
pg_script_body_Core.cpp (full scenario-based variables and callBacks)
pg_shader_body_bind_Core.cpp (allocates tables for passing C++ variable values to shaders)
```

    - *PG_glsl_source_generator* produces a new version of the scenes in which variables are ranked in the same order as in the full scenarios

    - *PG_glsl_source_generator* generates glsl shader code (full versions) to bind the C++ variables with glsl variables in the shaders (used for generating new full shaders when the shader codes are modified)

It is not necessary to use these generators before executing porphyrograph if the full scenario and the shaders are not modified

- The C++ modules have been reorganized by finer grained functionalities, all global variables and functions have been renamed with a pg_ prefix. Enumerations have been prefixed with pg_enum_. The modules are

```
# global includes
include/pg-all_include.h
# audio processing (PureData, PortAudio)
include/pg-audio.h
pg-audio.cpp
# generic callBack functions
include/pg-callBack.h
pg-callBack.cpp
# SVG clipArt (NVIDIA NV_path_rendering)
include/pg-clipArt.h
pg-clipArt.cpp
# color management and audio-based color modulation
include/pg-color.h
pg-color.cpp
# flashes between layers
include/pg-flash.h
pg-flash.cpp
# GUI management
include/pg-gui.h
pg-gui.cpp
# constants
include/pg-header.h
```

```
# variable initialization
include/pg-init.h
pg-init.cpp
# light management
include/pg-light.h
pg-light.cpp
# main function and GLUT call backs
include/pg-main.h
pg-main.cpp
# mesh managements (augmented reality or 3D rendering)
include/pg-mesh.h
pg-mesh.cpp
# messages display
include/pg-messages.h
pg-messages.cpp
# movie and camera streaming
include/pg-movieCamera.h
pg-movieCamera.cpp
# paths record/replay/save
include/pg-path.h
pg-path.cpp
# still image (diaporamas or animated clips)
include/pg-photoClip.h
pg-photoClip.cpp
# multipass GPU rendering
include/pg-render.h
pg-render.cpp
# dynamic variable management through scenes
include/pg-scenarioDyn.h
pg-scenarioDyn.cpp
# multi-scene scenario and resources parsing
include/pg-scenarioParse.h
pg-scenarioParse.cpp
# OSC message execution
include/pg-script.h
pg-script.cpp
# sensor management
include/pg-sensor.h
pg-sensor.cpp
# shader uniforms
include/pg-shader.h
pg-shader.cpp
# texture and FBO loading and allocating
include/pg-texture.h
pg-texture.cpp
# OSC/UDP messages reception/emission
include/pg-udp.h
pg-udp.cpp
# C++ glue
pg-template.hpp
```