GPU TECHNOLOGY CONFERENCE

BRINGING PHYSICALLY BASED RENDERING TO YOUR APPLICATION

MARTIN-KARL LEFRANÇOIS, DEVTECH, NVIDIA ARC

INTRODUCTION

- Overview of NVIDIA® Iray®
- Integration 101 of NVIDIA Iray
  - Initialization
  - Creating a scene: camera, lights, materials, geometry
  - Rendering techniques
  - Debugging
  - Resources

WHAT IS IRAY

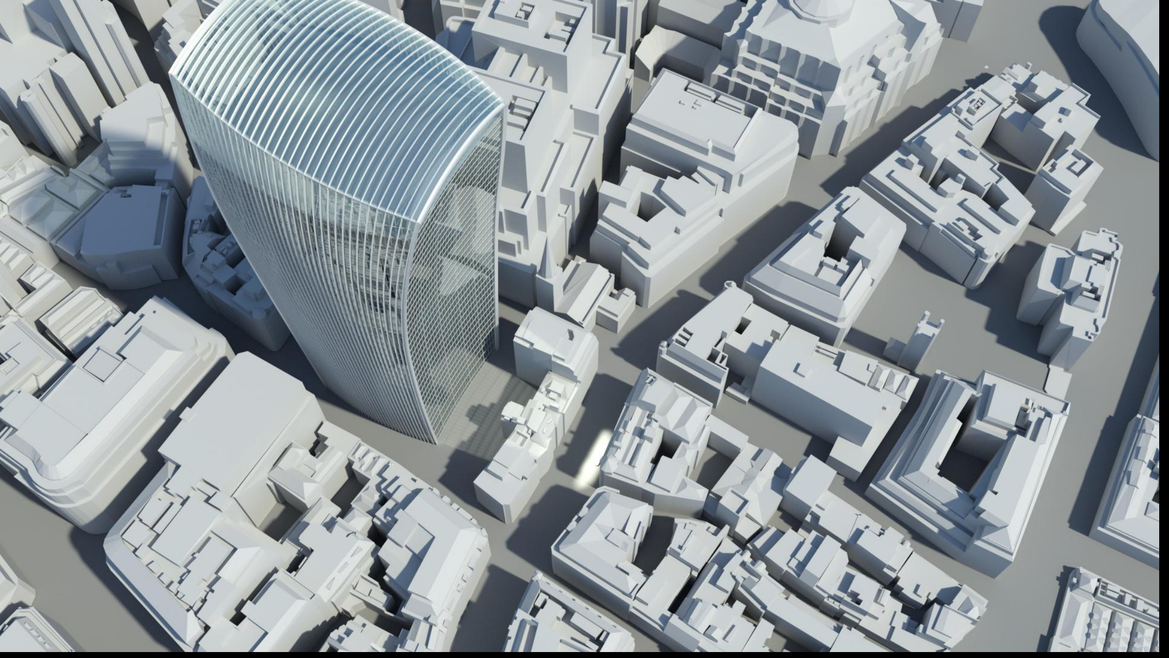
- Physically based push button rendering solution
- Multiple rendering modes for interactive to final frame workflows
- One unified Material Definition Language (MDL)
- Cluster and cloud rendering with efficiency in scaling
- SDK for system integration

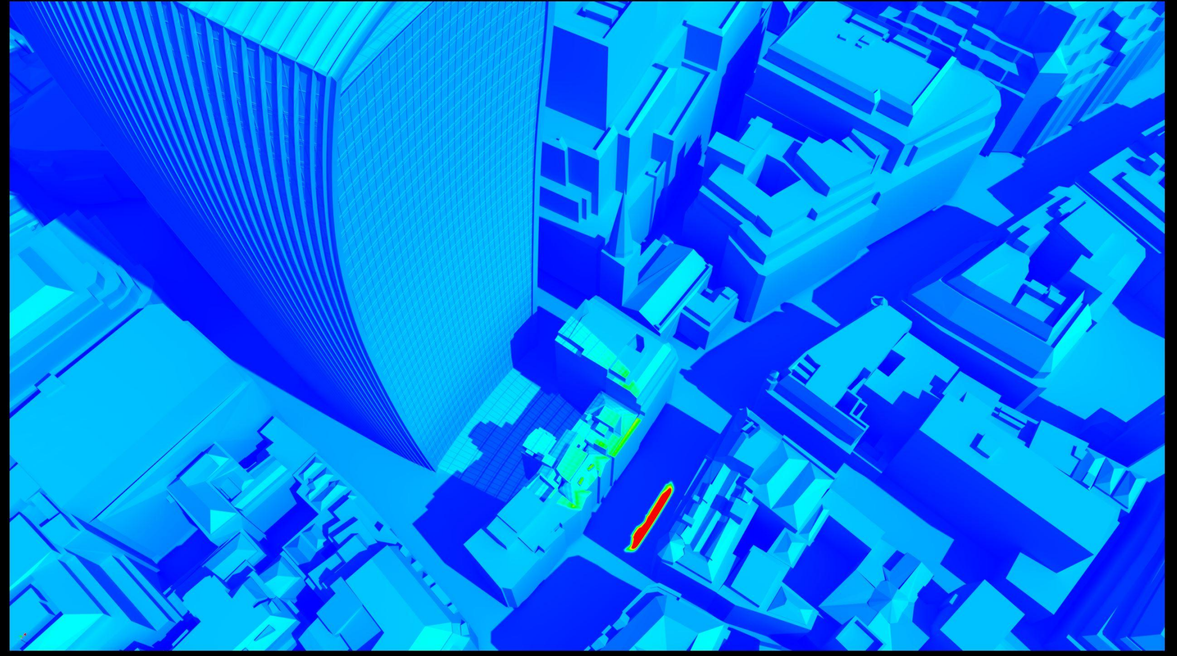

Bunkspeed

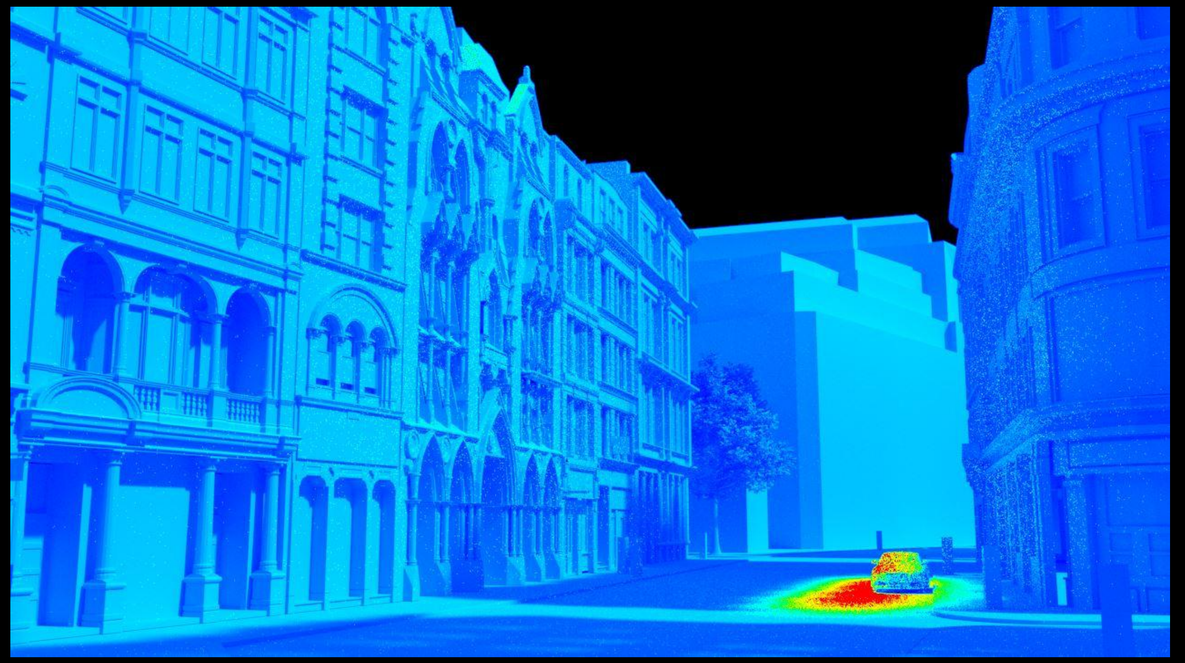

B - TZ 2012


Isabelle Lancray
PARIS
L'ÂGE D'OR
MARIANNE
CREME LIBERTE
ETERNAL CREAM
50ml - 1.7 FL.OZ
Isabelle Lancray
PARIS
L'ÂGE D'OR
EDITH
CREME ABSOLUE ABSOLUTE CREAM
Isabelle Lancray
PARIS

Bunkspeed





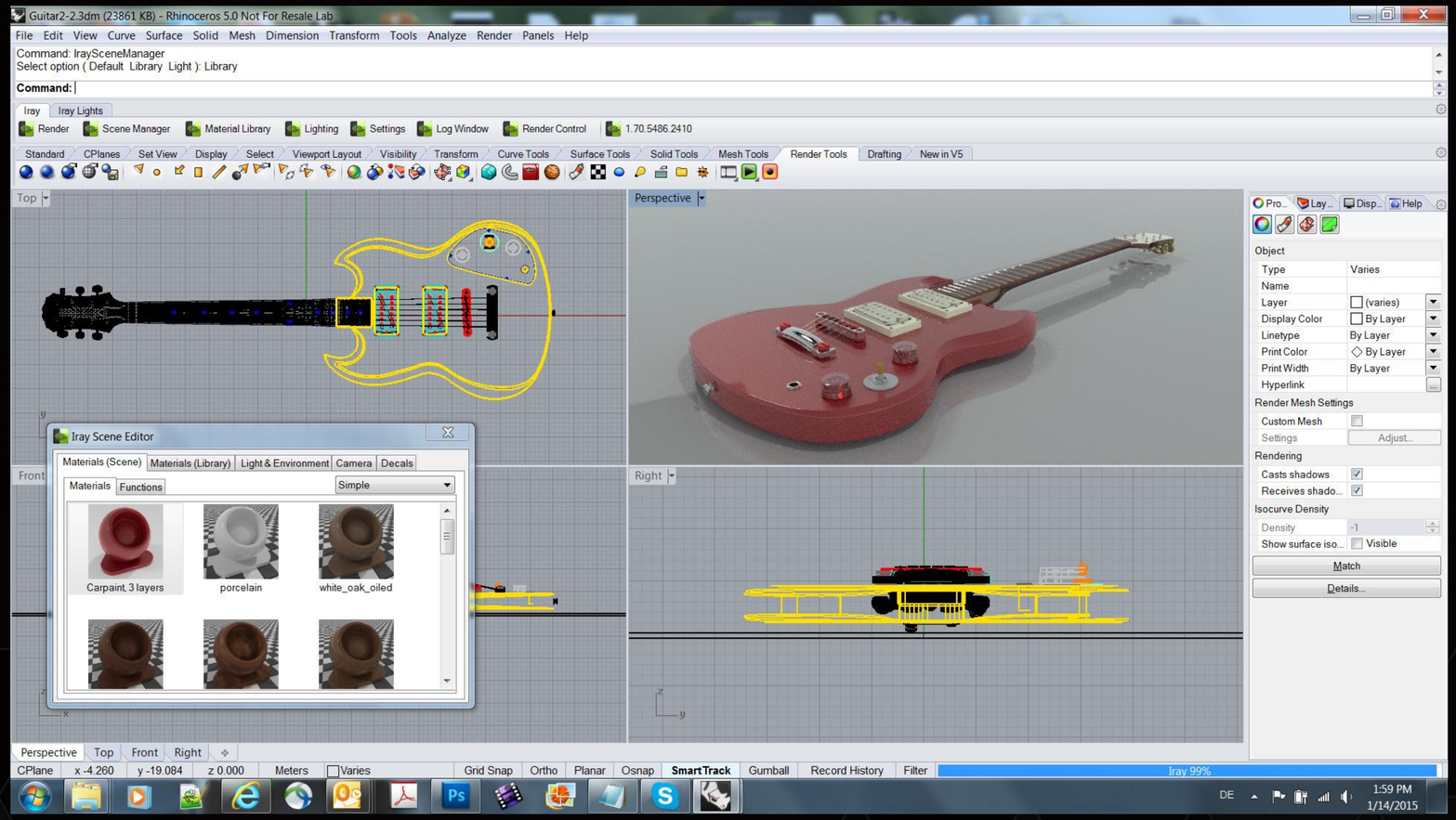


INTEGRATION INTO PRODUCTS

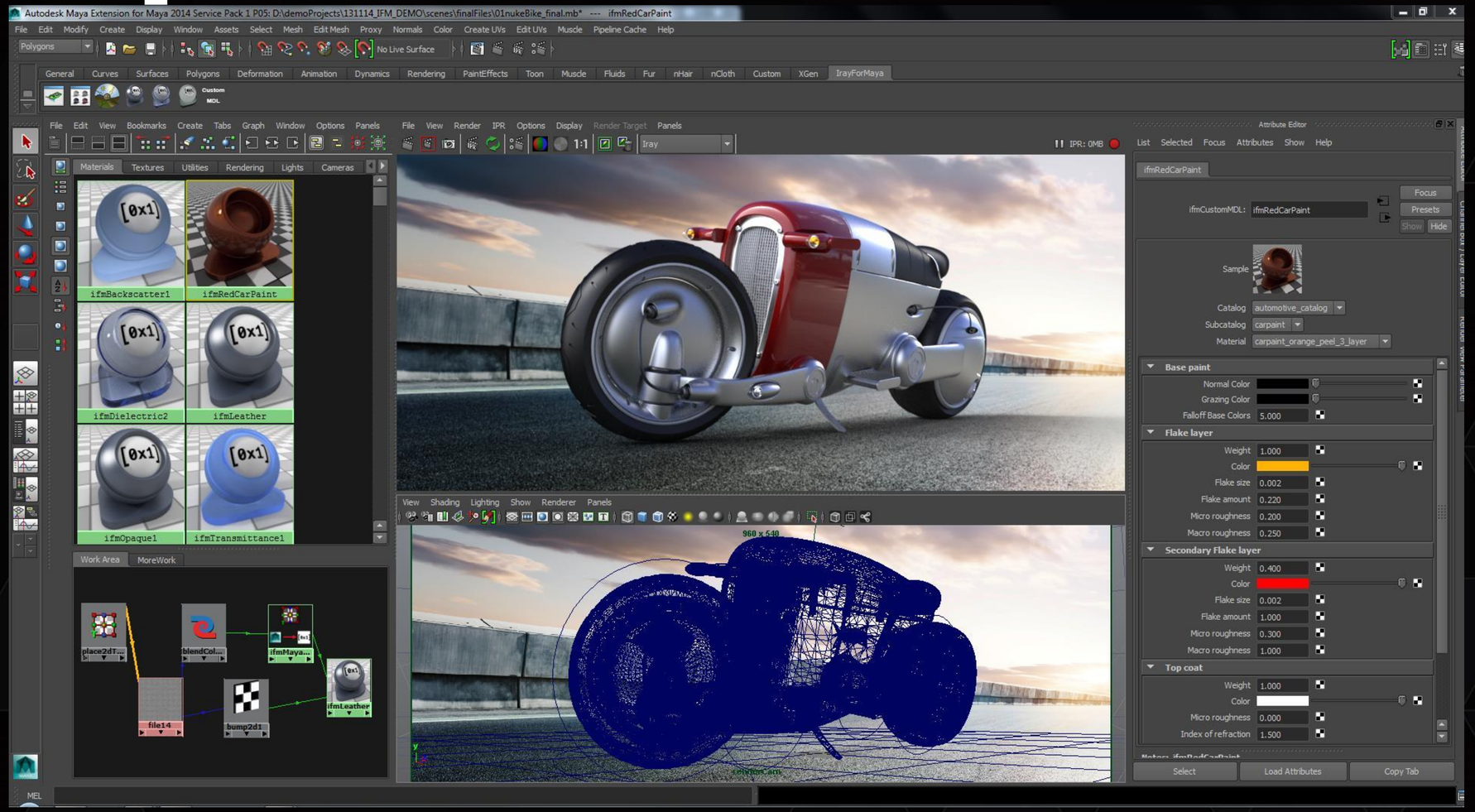
BUNKSPEED

DASSAULT: CATIA LIVE RENDERING

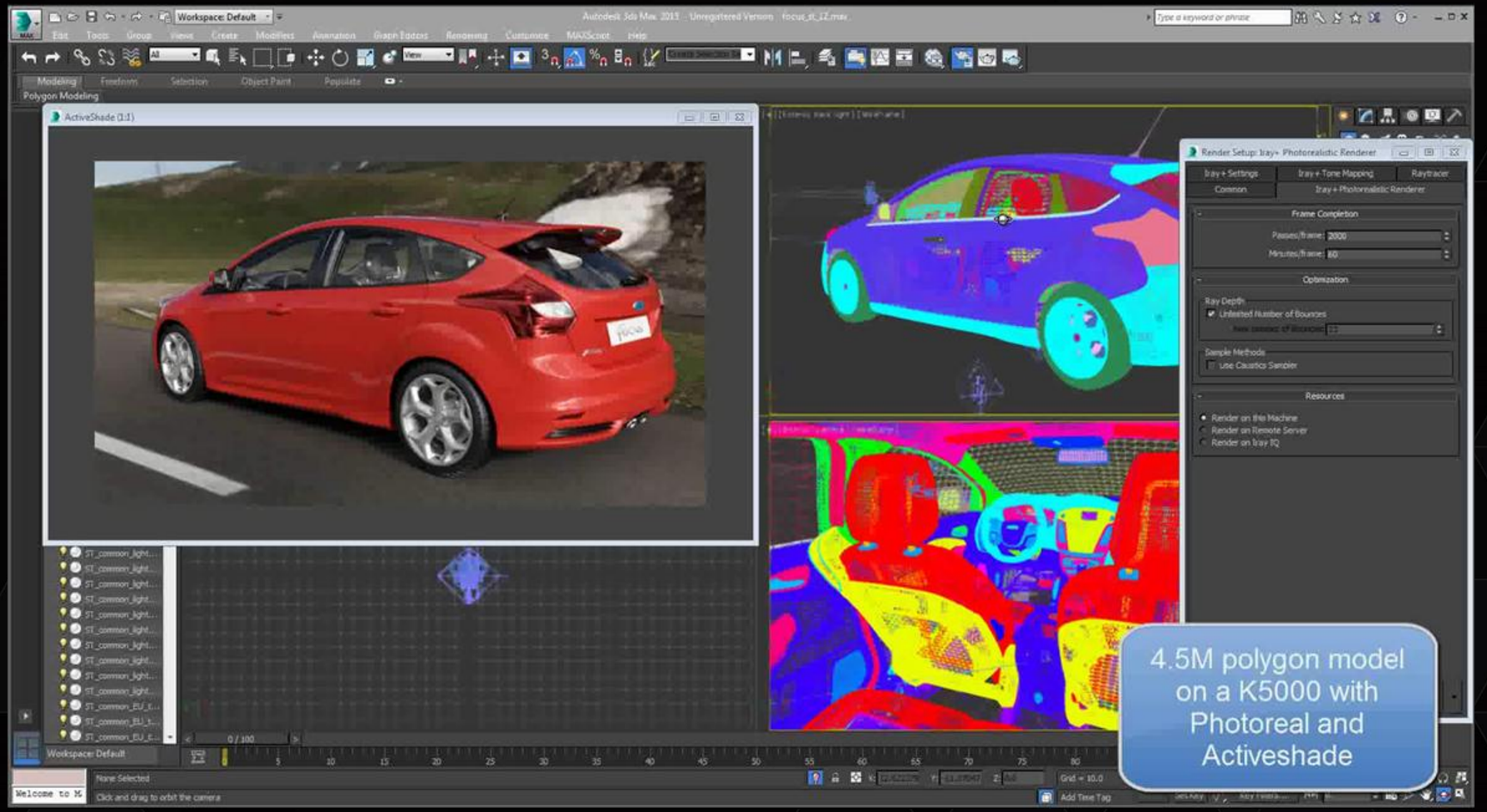
RHINO



[0X1]: IRAY FOR MAYA



LIGHTWORKS: 3DSMAX



GPU TECHNOLOGY CONFERENCE

DAZ3D

DAZ Studio software

- Millions of downloads – helps fuel our robust user community
- Offered *free* since 2012
- Major upgrades in 2015 put Studio on par with leading 3D programs

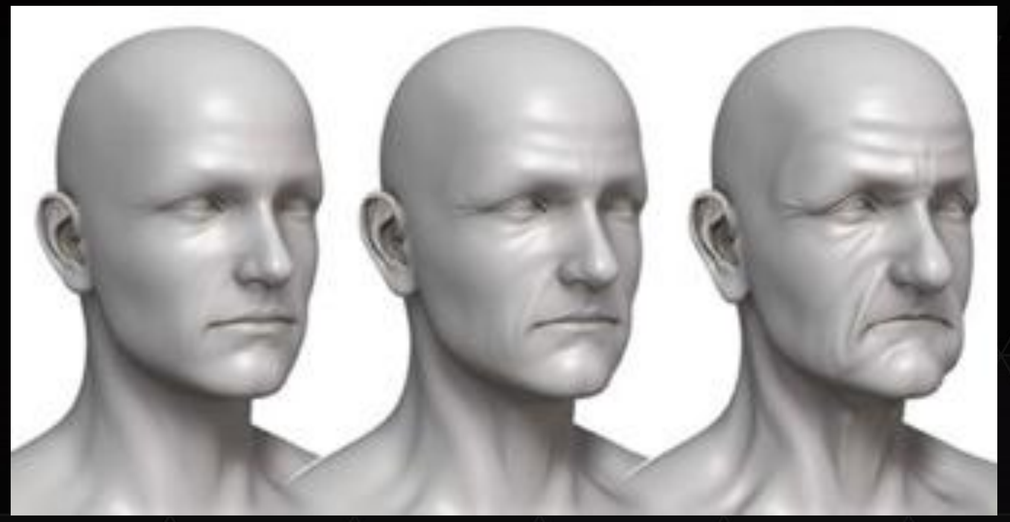
Engaged Community

- Site traffic resulting in ~200,000,000 pageviews per year
- Avg. time spent per visit at daz3d.com (Q4'14) = *21 minutes*
- Hundreds of Artists release new 3D content for sale every month

GENESIS 2

With DAZ's proprietary Genesis figure platform in DAZ Studio, human models can be 'morphed' in a nearly endless variety of ways – like body shape, size, type, age, etc.

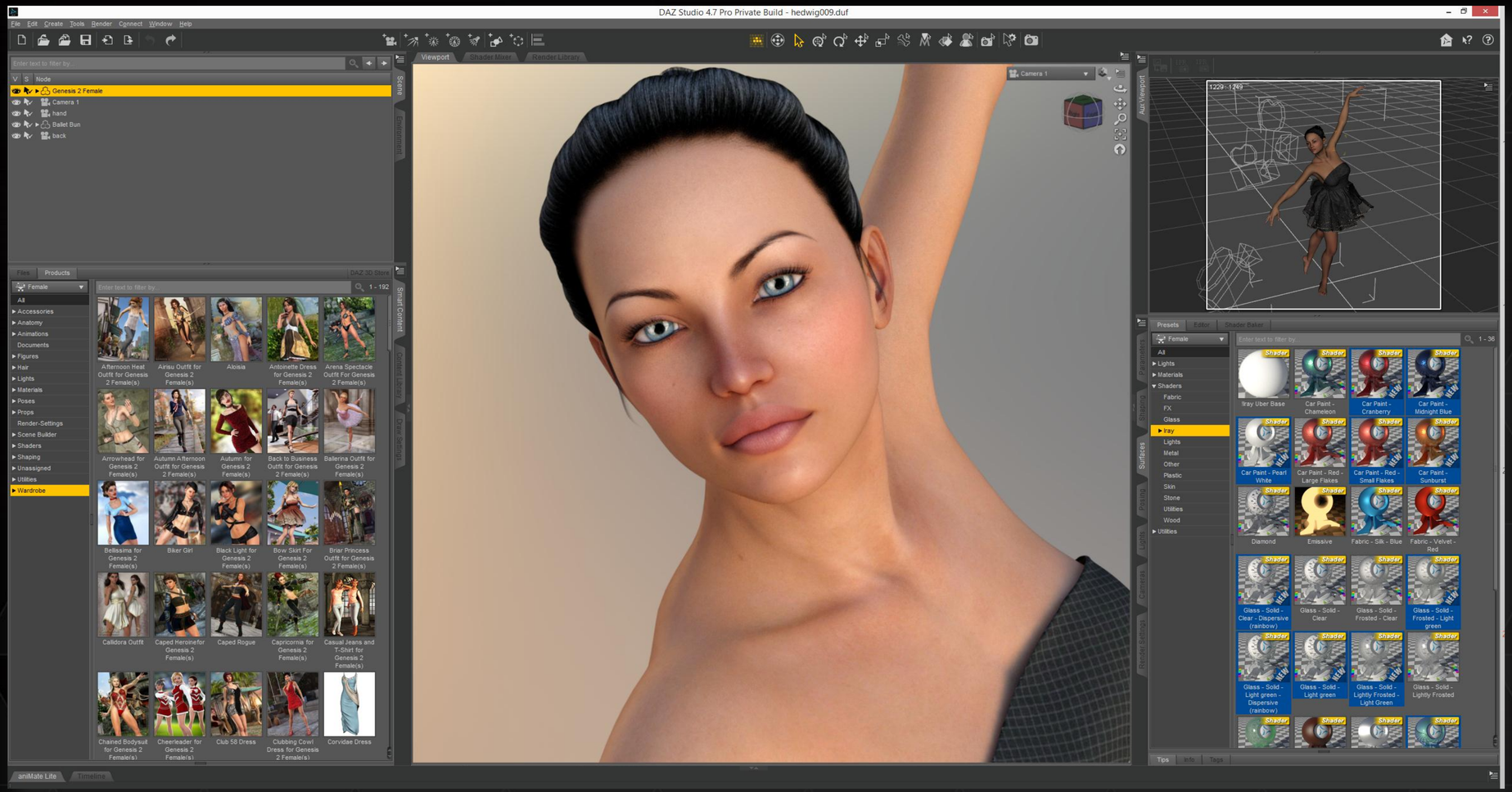
Customers can do this with a series of simple sliders and buttons in DAZ Studio.



GPU TECHNOLOGY CONFERENCE

DAZ STUDIO

IRADIUM



DAZ Studio 4.7

DAZ Studio Iradium

DAZ Studio 4.7

DAZ Studio Iradium

DAZ Studio - 3Delight

DAZ Studio - Iray 2015

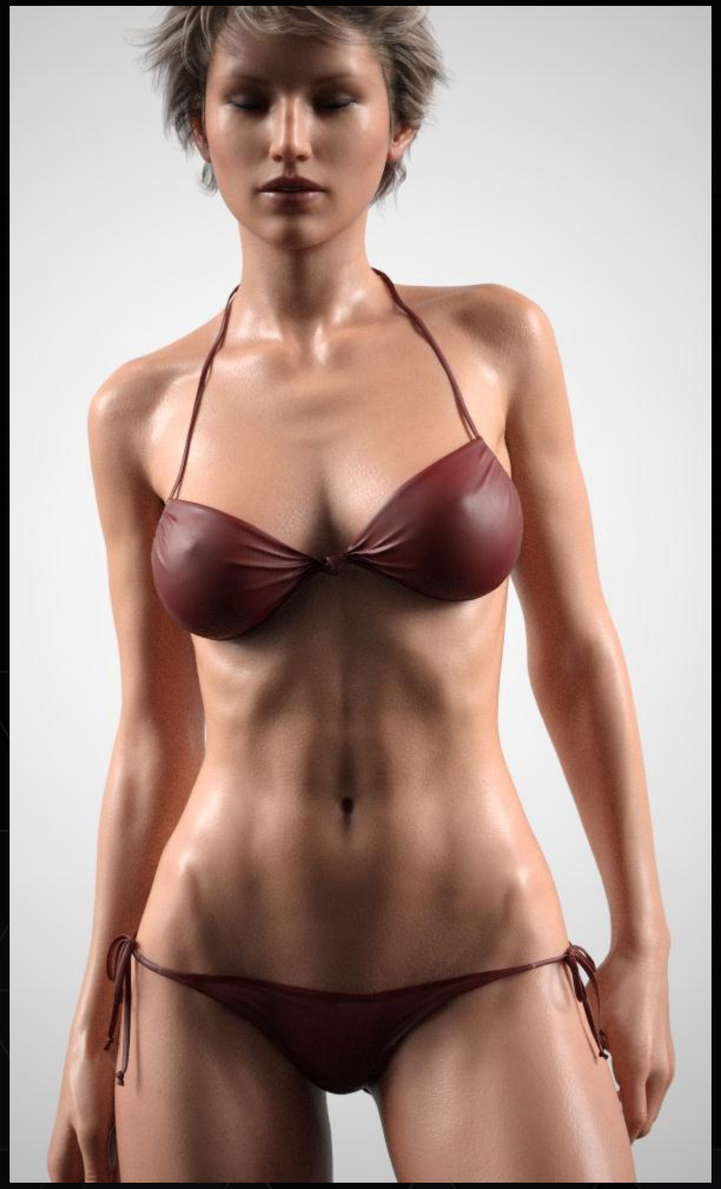
LuxRender - 10h

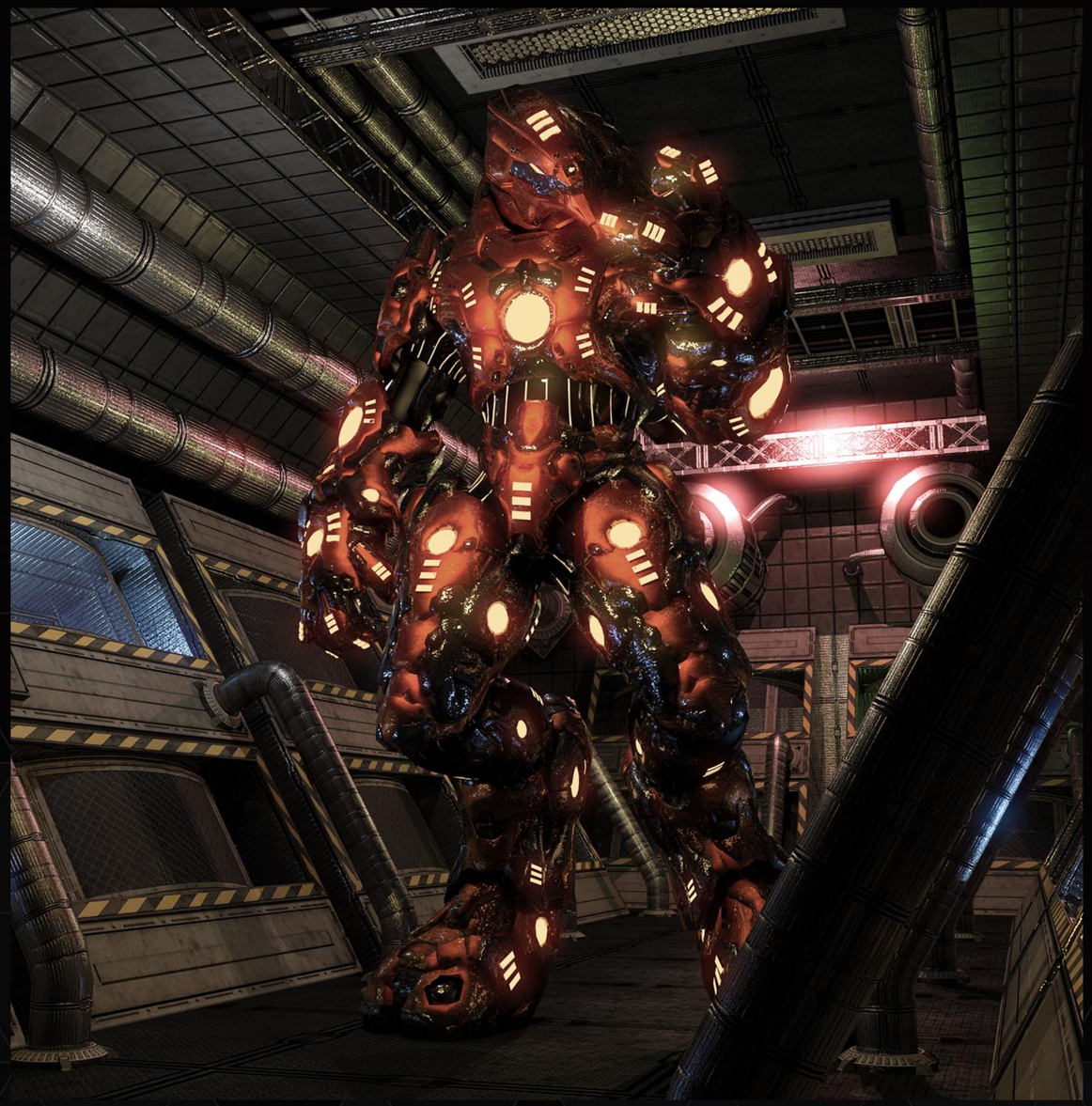
Iray 2015 – 4h10

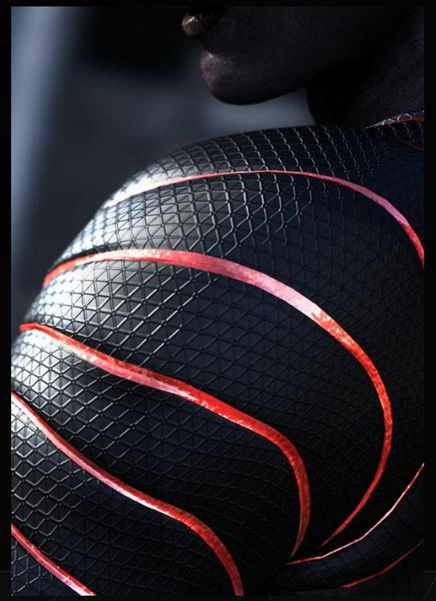
CPU only render on MAC Pro Dual Quad-Core 2.8GHz

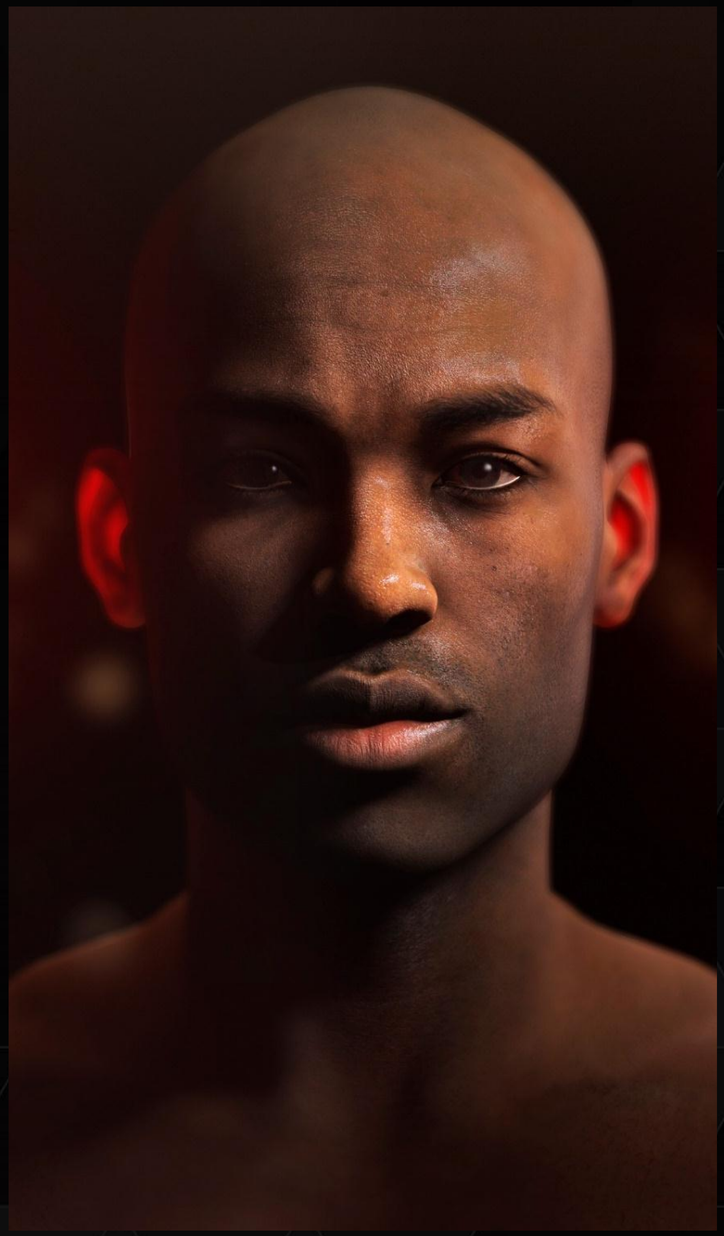
LuxRender - 10h

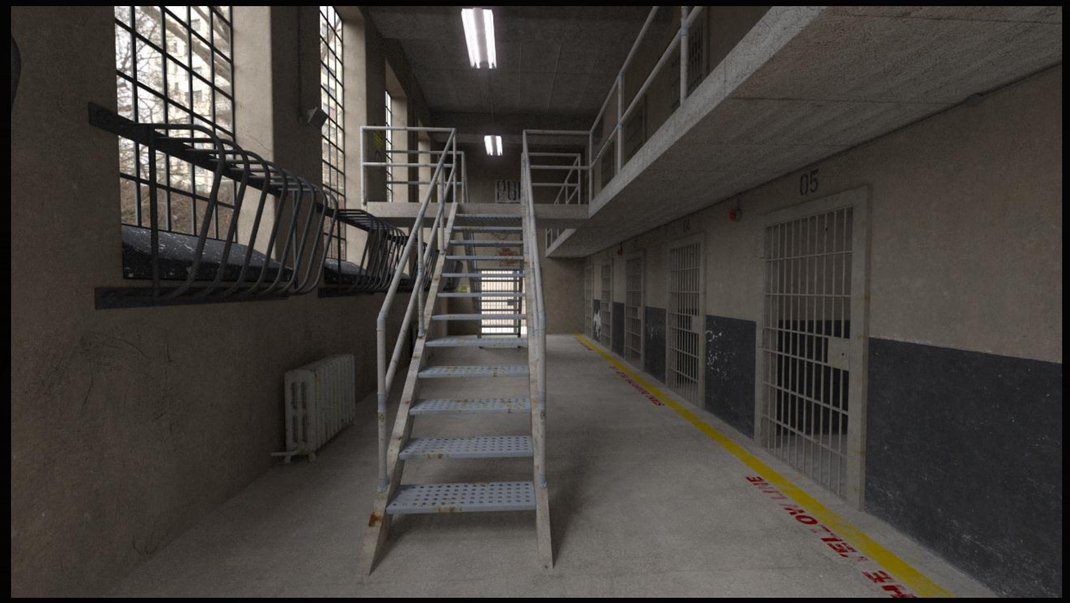
Iray 2015 – 4h10

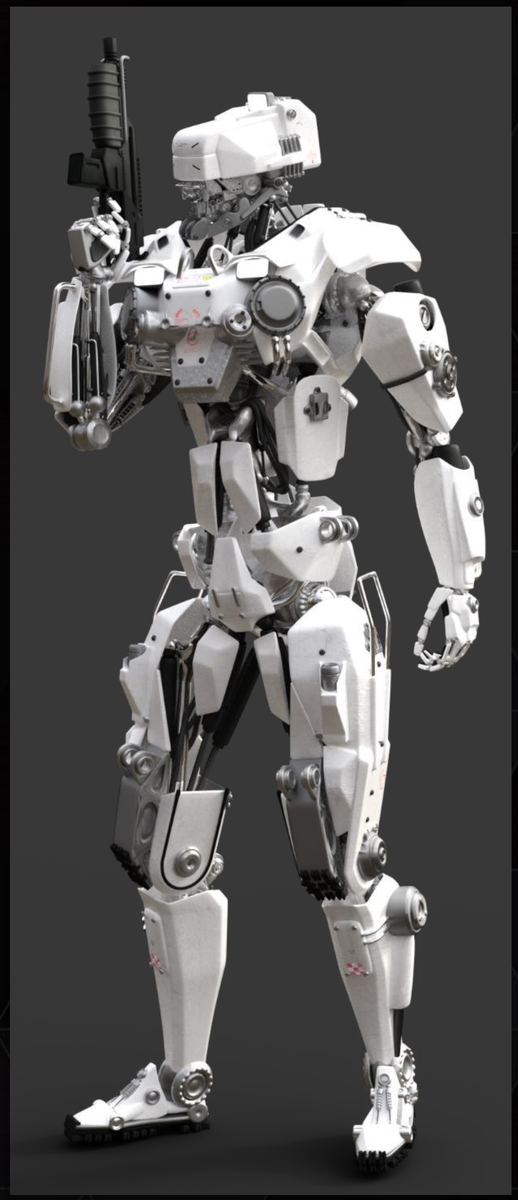












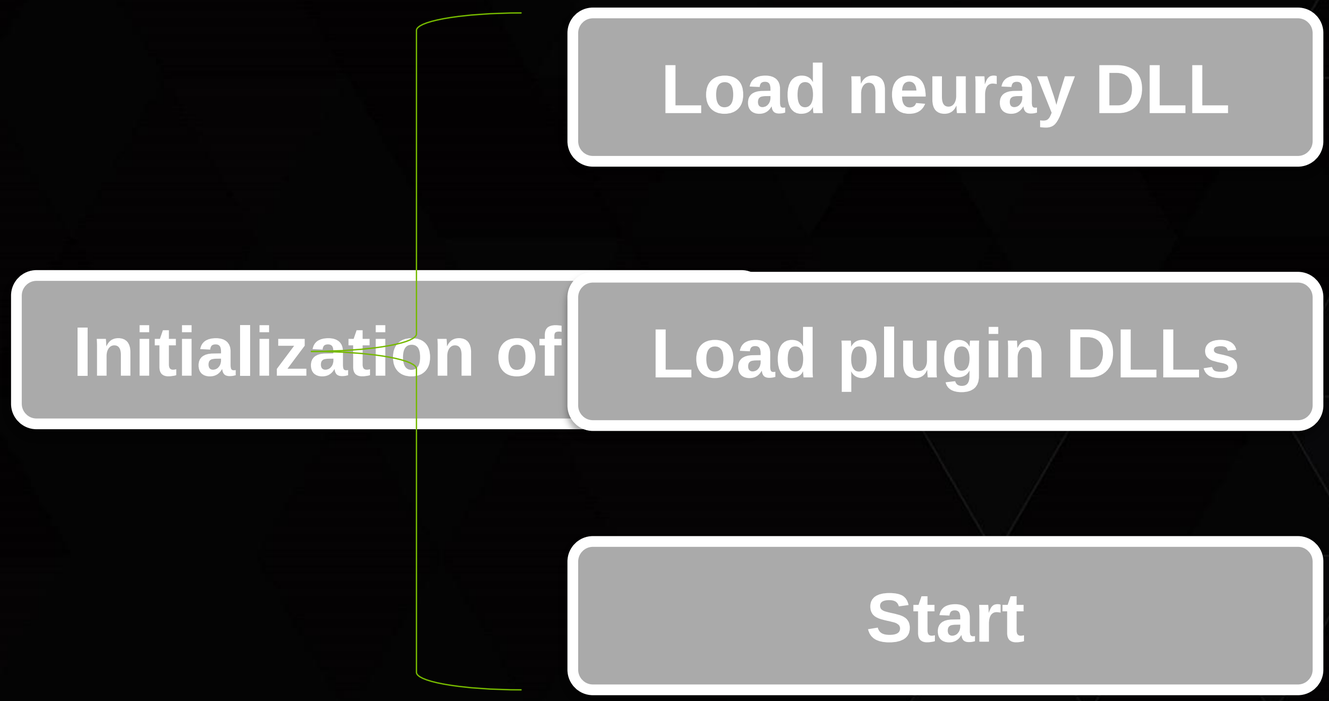
INTEGRATING IRAY 101

Initialization of Iray

Create Scene

Rendering Images

INITIALIZING IRAY



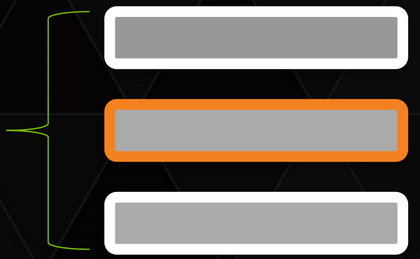
LOADING NEURAY DLL

```
void* handle = LoadLibraryA((LPSTR)"libneuray.dll");
void* symbol = GetProcAddress((HMODULE)handle, "mi_neuray_factory");

typedef mi::neuraylib::INeuray* (INeuray_factory)(mi::neuraylib::IAllocator*, mi::Uint32);
INeuray_factory* factory = (INeuray_factory*)symbol;

mi::neuraylib::INeuray* neuray = factory(0, MI_NEURAYLIB_API_VERSION);
```

INeuray: is the root to access everything

LOADING PLUGINS

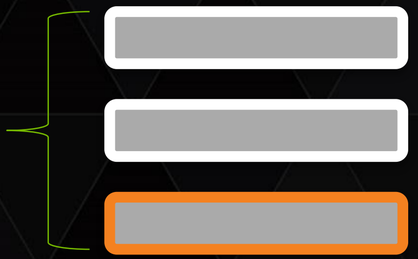
```
{
LOAD PLUGINS: FREEIMAGE FOR TEXTURES AND IRAY PHOTOREAL RENDERER
    mi::base::Handle<mi::neuraylib::IPlugin_configuration> plugin_configuration(
        neuray->get_api_component<mi::neuraylib::IPlugin_configuration>());
    plugin_configuration->load_plugin_library("freeimage.dll");
    plugin_configuration->load_plugin_library("libiray.dll");
}
```



STARTING Iray

START UP NEURAY.

```
int error_code = neuray->start(true);
```

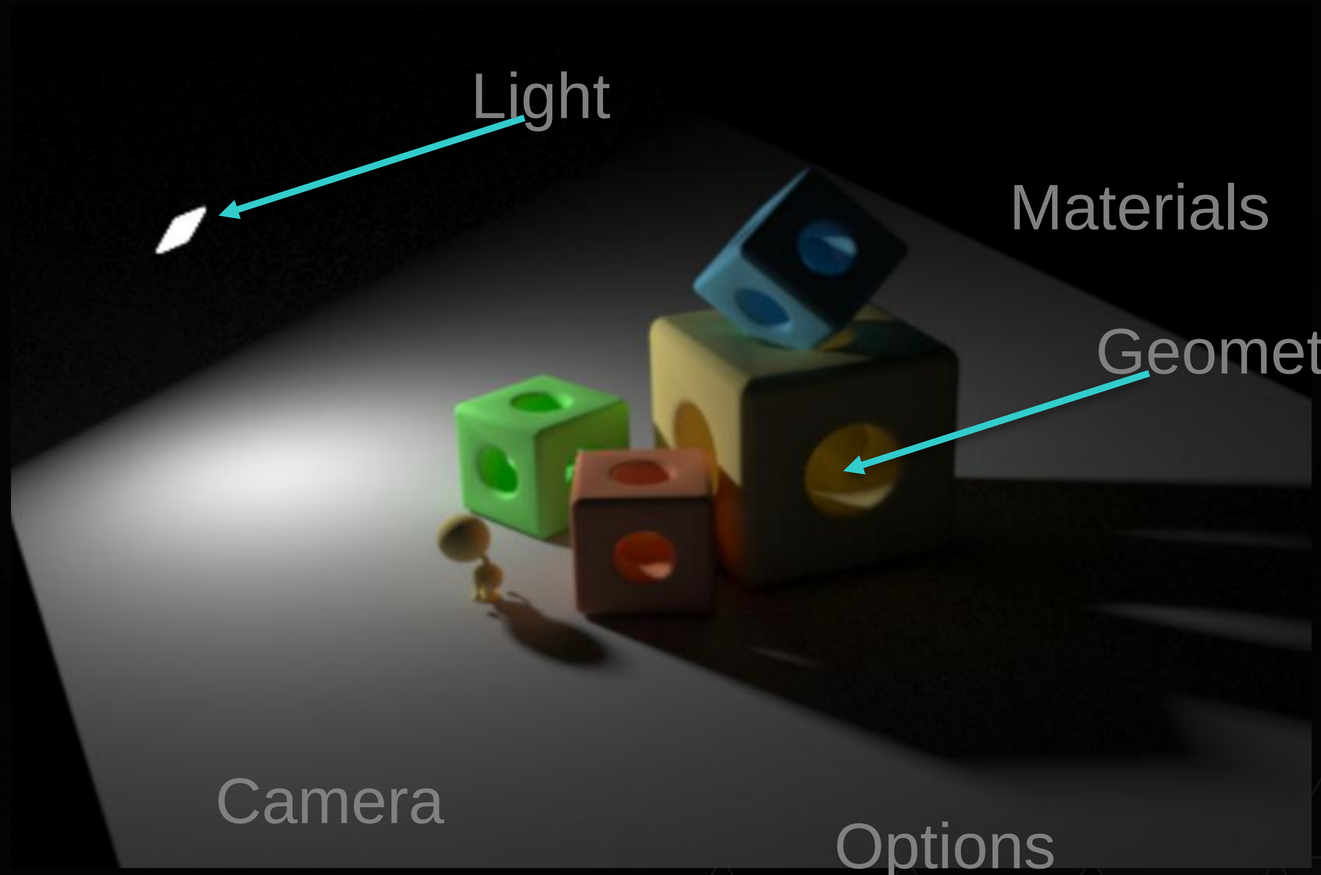


INTEGRATING IRAY 101

Initialization of Iray

Create Scene

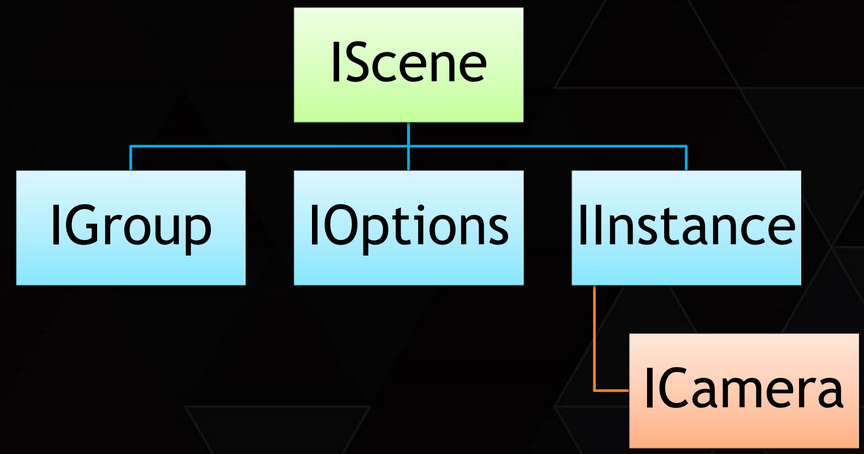
CREATING THE SCENE



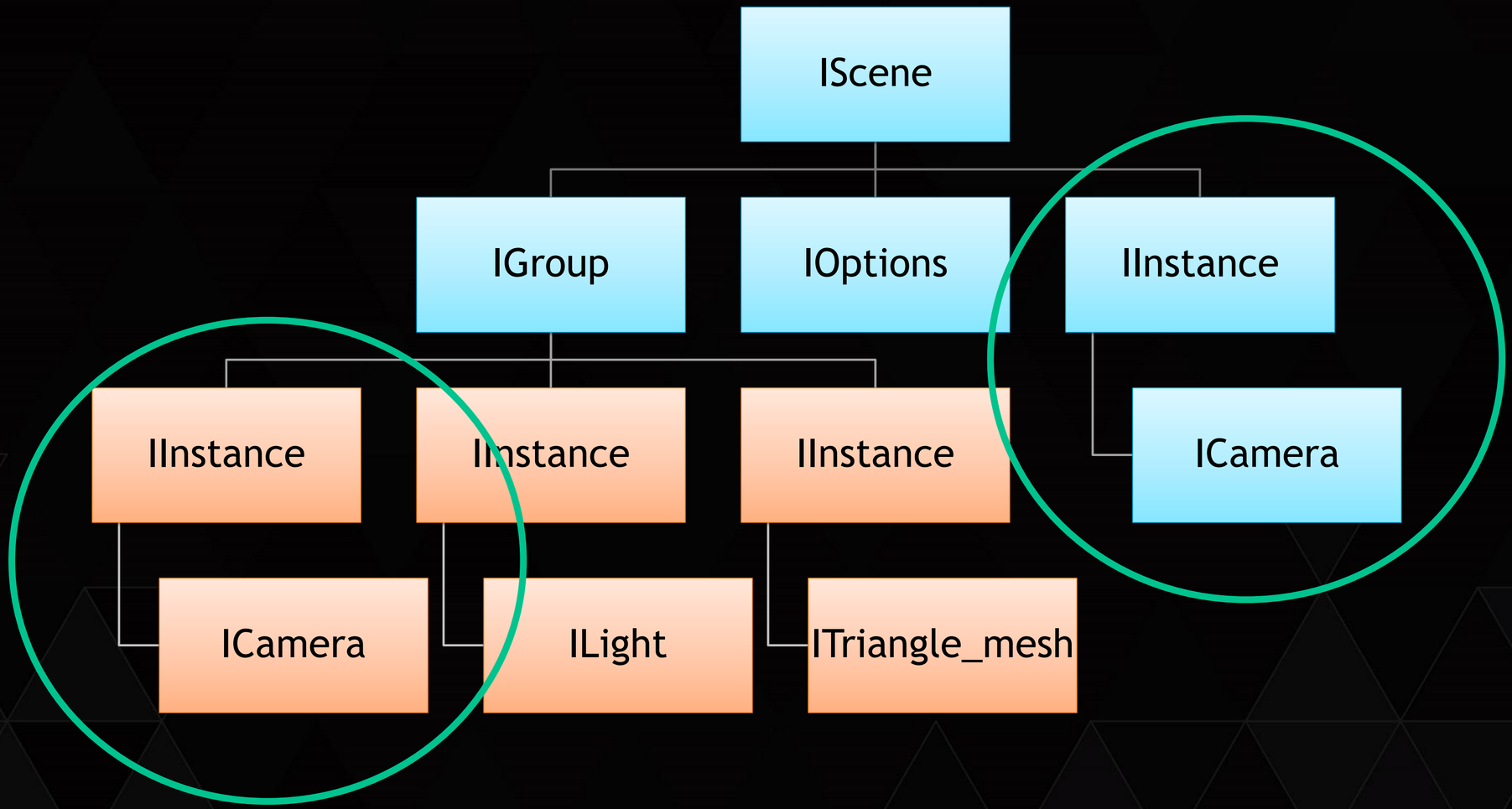
SCENE

- Top-level element describing a subset of DB elements to be rendered.

- A scene is described by three elements:
  - Root group:
    - which is the root node of the scene graph of type mi::neuraylib::IGroup
  - Options:
    - The global scene options of type mi::neuraylib::IOptions
  - Camera:
    - The camera instance of type mi::neuraylib::IInstance



SCENE HIERACHY



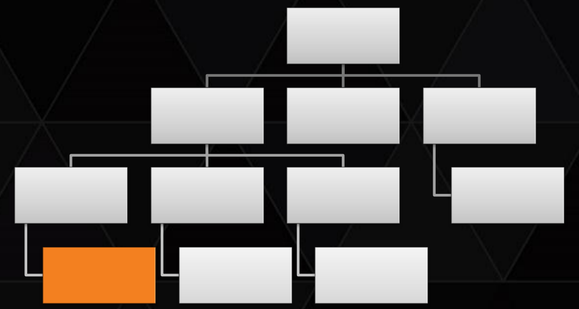
TRANSACTIONS

- Provides a consistent view on the database.
- Isolated from changes by other (parallel) transactions.
- Each transaction must be either committed or aborted, i.e., all changes become either atomically visible to transactions started afterwards, or not at all.

```
mi::base::Handle<mi::neuraylib::ITransaction> transaction(neuray.create_transaction());
```

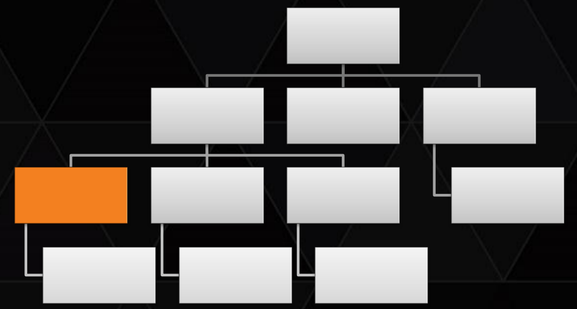
CAMERA

```
{
CREATE A CAMERA
    Handle<ICamera> camera(transaction->create<ICamera>("Camera"));
    camera->set_orthographic(false);
    camera->set_focal(50.f);
    camera->set_aperture(44.f);
    camera->set_resolution_x(640);
    camera->set_resolution_y(480);
    camera->set_aspect(640.f / 480.f);

STORE CAMERA IN DATABASE
    transaction->store(camera.get(), "camera");
}
```



CAMERA INSTANCE

```
{
CREATING THE CAMERA INSTANCE
    Handle<IInstance> instance(transaction->create<IInstance>("Instance"));
    instance->attach("camera");

SETTING THE POSITION AND ORIENTATION OF THE CAMERA
    mi::Float64_4_4 matrix(1);
    matrix.lookat(mi::Float32_3(0, 2, 4), mi::Float32_3(0, 0, 0), mi::Float32_3(0, 1, 0));
    instance->set_matrix(matrix);

    transaction->store(instance.get(), "camera_instance");
}
```



MATERIALS (MDL)

The Material Definition Language MDL

- is technology developed by NVIDIA
- to define physically-based materials
- for its advanced rendering solutions
  - NVIDIA Iray®
  - mental ray®
- central for physically- based rendering

MATERIALS

GRAY MATERIAL

```
mi::nrx::Mdl matGray(transaction, neuray.factory(),
        "mdl::main::diffuse_material", "mat_gray");
matGray>set_value("tint", to_linear(mi::Spectrum(0.6f, 0.6f, 0.6f)));
```

WHITE LIGHT

```
mi::nrx::Mdl matLight(transaction, neuray.factory(),
        "mdl::main::diffuse_light", "mat_light");
matLight->set_value("tint", mi::Spectrum(1000.0f, 1000.0f, 1000.0f));
```

COMPENSATE FOR GAMMA SPACE

```
inline mi::Spectrum to_linear(const mi::Spectrum& color)
{
    static const float gamma = 2.2f;
    mi::Spectrum c( powf(color[0], gamma), powf(color[1], gamma),  powf(color[2], gamma));
    return c;
}
```

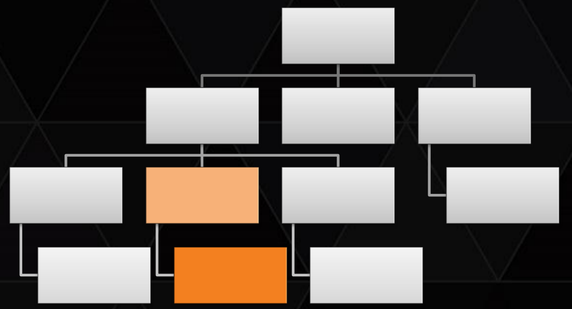
LIGHT

CREATE THE LIGHT "LIGHT”

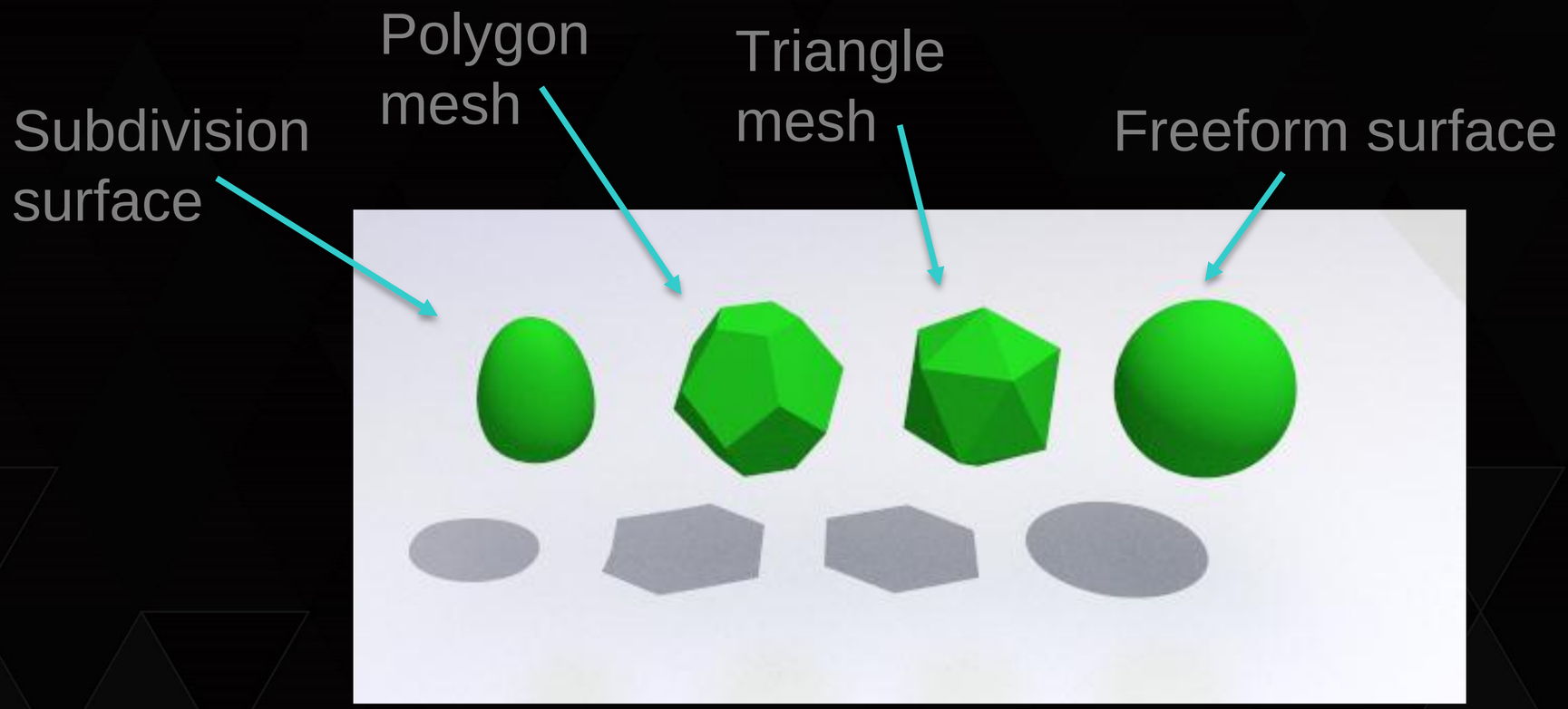
```
{
    Handle<ILight> light(transaction->create<ILight>("Light"));
    Handle<mi::IRef> material(light->create_attribute<mi::IRef>("material", "Ref"));
    material->set_reference("mat_light");
    transaction->store(light.get(), "light");
}
```

CREATE THE INSTANCE "LIGHT_INSTANCE" REFERENCING "LIGHT”

```
{
    Handle<IInstance> instance(transaction->create<IInstance>("Instance"));
    instance->attach("light");
    mi::Float64_4_4 matrix(1.0f);
    matrix.translate(-3.0f, -8.5f, -3.0f); // World to object space
    instance->set_matrix(matrix);
    transaction->store(instance.get(), "light_instance");
}
```



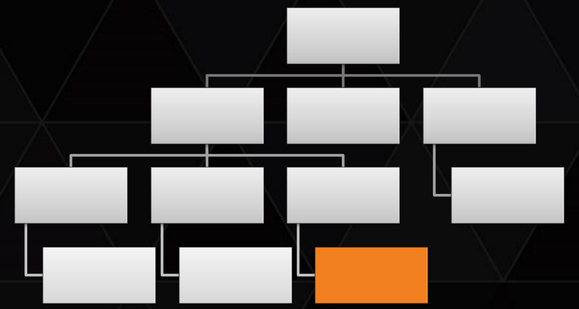
GEOMETRY



GEOMETRY

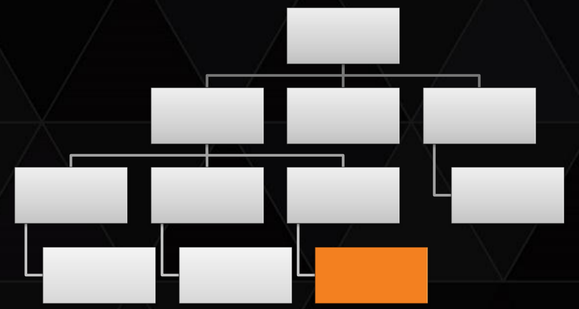
CREATING A GROUND PLANE, MADE OF TWO TRIANGLES

```
{
    // Geometry of the grey ground plane.
    const mi::Uint32 ground_n_points    = 4;
    const mi::Uint32 ground_n_normals   = 1;
    const mi::Uint32 ground_n_triangles = 2;

    mi::Float32_3 ground_points[ground_n_points] ={
            mi::Float32_3(-2.0, 0.0, -2.0),
            mi::Float32_3(-2.0, 0.0, 2.0),
            mi::Float32_3(2.0, 0.0, -2.0),
            mi::Float32_3(2.0, 0.0, 2.0)};

    mi::Float32_3 ground_normals[ground_n_normals] ={
            mi::Float32_3(0.0, 1.0, 0.0)};
```



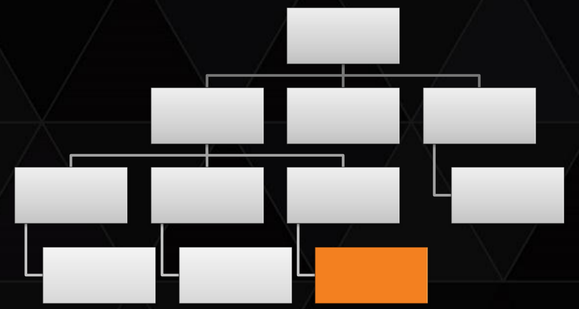
GEOMETRY - 2

```
TRIANGLE INDICES
  Triangle_point_indices ground_mesh_connectivity[ground_n_triangles] ={
        Triangle_point_indices(0, 1, 3),
        Triangle_point_indices(3, 2, 0) };

CREATE THE TRIANGLE MESH
  Handle<ITriangle_mesh> mesh(transaction->create<ITriangle_mesh>("Triangle_mesh"));

ADD ALL MESH POINTS
  for (mi::Uint32 i = 0; i < ground_n_points; ++i)
    mesh->append_point(ground_points[i]);

ADD ALL MESH TRIANGLES
  for (mi::Uint32 i = 0; i < ground_n_triangles; ++i)
    mesh->append_triangle(ground_mesh_connectivity[i]);
```



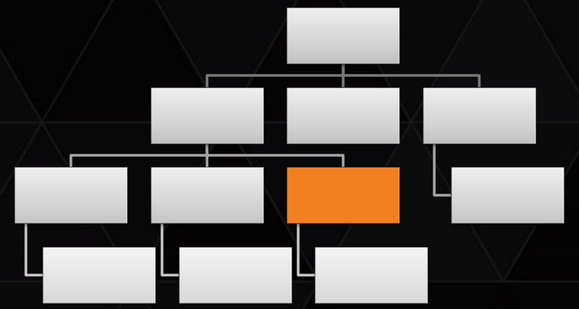
GEOMETRY - 3

```
CREATE CONNECTIVITY: HOW NORMAL ARE CONNECTED TO TRIANGLES
    Handle<ITriangle_connectivity> connectivity(mesh->create_connectivity());

CREATE NORMAL ATTRIBUTES
    Handle<IAttribute_vector> normals(
                    connectivity->create_attribute_vector(mi::neuraylib::ATTR_NORMAL));
    normals->append_vector3(ground_normals[0]);
    connectivity->attach_attribute_vector(normals.get());;

NORMAL INDEX PER TRIANGLE POINT
    Triangle_point_indices ground_normal_connectivity[ground_n_triangles] ={
            Triangle_point_indices(0, 0, 0),
            Triangle_point_indices(0, 0, 0)};

 SET NORMAL INDICES TO EACH TRIANGLE
    for (mi::Uint32 i = 0; i < ground_n_triangles; ++i)
            connectivity->set_triangle_indices(Triangle_handle(i), ground_normal_connectivity[i]);
    mesh->attach_connectivity(connectivity.get());;

    transaction->store(mesh.get(), "ground");
}
```



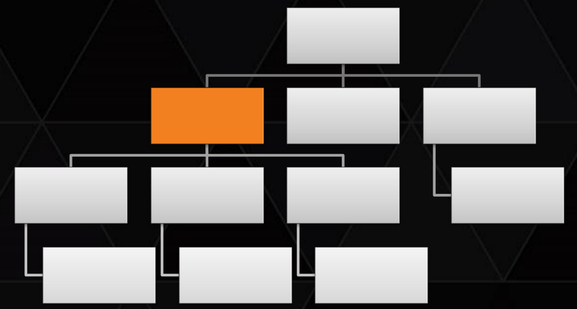
GEOMETRY INSTANCE

```
{
CREATE THE INSTANCE "GROUND_INSTANCE" REFERENCING "GROUND”
    Handle<IInstance> instance(transaction->create<IInstance>("Instance"));
    instance->attach("ground");

ATTACH THE MATERIAL
    Handle<mi::IRef> material(instance->create_attribute<mi::IRef>("material", "Ref"));
    material->set_reference("mat_gray");

STORE THE INSTANCE
    transaction->store(instance.get(), "ground_instance");
}
```



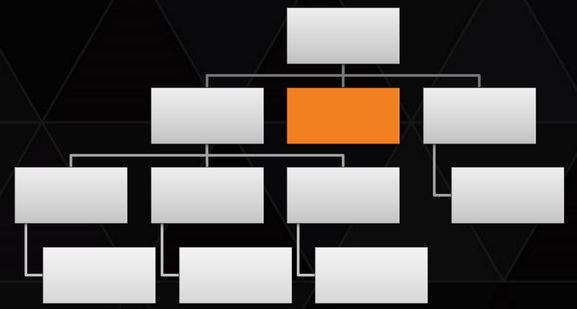
ROOT GROUP

```
{
THIS IS THE ROOT GROUP, WHICH IS NEEDED TO RENDER THE SCENE
    Handle<IGroup> group(transaction->create<IGroup>("Group"));
    group->attach("camera_instance");
    group->attach("light_instance");
    group->attach("ground_instance");
    group->attach("cube_instance");

MAKE ALL ELEMENTS VISIBLE
    Handle<mi::IValue> attribute(group->create_attribute<mi::IValue>("visible", "Boolean"));
    attribute->set_value(true);

    transaction->store(group.get(), "root_group");
}
```



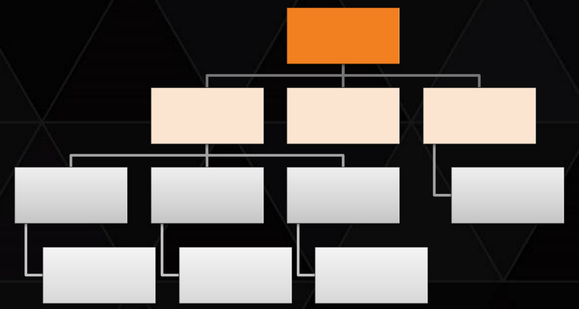
SCENE OPTIONS

```
{
CREATE THE SCENE OPTIONS
    Handle<IOptions> options(transaction->create<IOptions>("Options"));

STOP RENDERING AFTER 25 ITERATIONS
    Handle<mi::ISint32> prs(
                    options->create_attribute<mi::ISint32>(
                                "progressive_rendering_max_samples", "Sint32"));
    prs->set_value(25);

    transaction->store(options.get(), "options");
}
```



SCENE ELEMENT

```
{
CREATE THE SCENE OBJECT
  Handle<IScene> scene(transaction->create<IScene>("Scene"));
  scene->set_rootgroup("root_group");
  scene->set_options("options");
  scene->set_camera_instance("camera_instance");

STORE THE SCENE OBJECT IN THE DATABASE
  transaction->store(scene.get(), "my_scene");
}

DONE WITH THE CHANGES, COMMIT THE TRANSACTION
  transaction->commit();
```



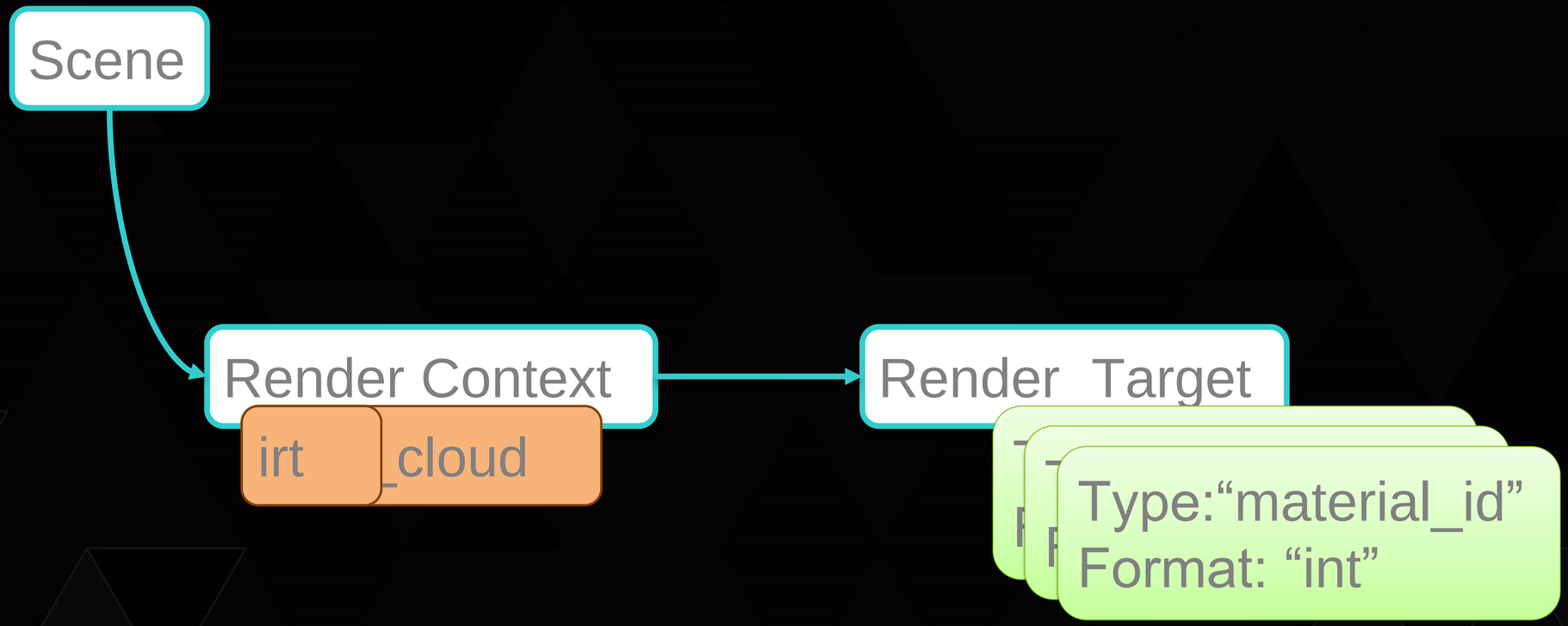
INTEGRATING IRAY 101

Initialization Iray

Create a Scene

Render Images

RENDERING



RENDER CONTEXT AND TARGET

THE RENDERING CONTEXT IS CREATED FROM THE SCENE

```
Handle<IScene> scene(transaction->edit<IScene>("my_scene");
Handle<IRender_context> render_context(
                        scene->create_render_context(transaction.get(), "iray"));
```

RENDERING TARGET (RENDERING BUFFER)

```
Handle<IImage_api> image_api(neuray->get_api_component<IImage_api>());
mi::nrx::Render_target* render_target =
                new mi::nrx::Render_target(image_api.get(), "result", "Rgba", 640, 480);
```

RENDERING

THIS A BLOCKING FUNCTION, ONCE OUT, THE FRAME IS RENDERED

```
{
    mi::base::Handle<ITransaction> transaction(neuray.create_transaction());
    render_context->render(transaction.get(), render_target, nullptr);
    neuray.commit_transaction(transaction);
}
```

SAVING IMAGE

GET THE RENDERED IMAGE

```
mi::base::Handle<mi::neuraylib::ICanvas> canvas(render_target->get_canvas(0));
```

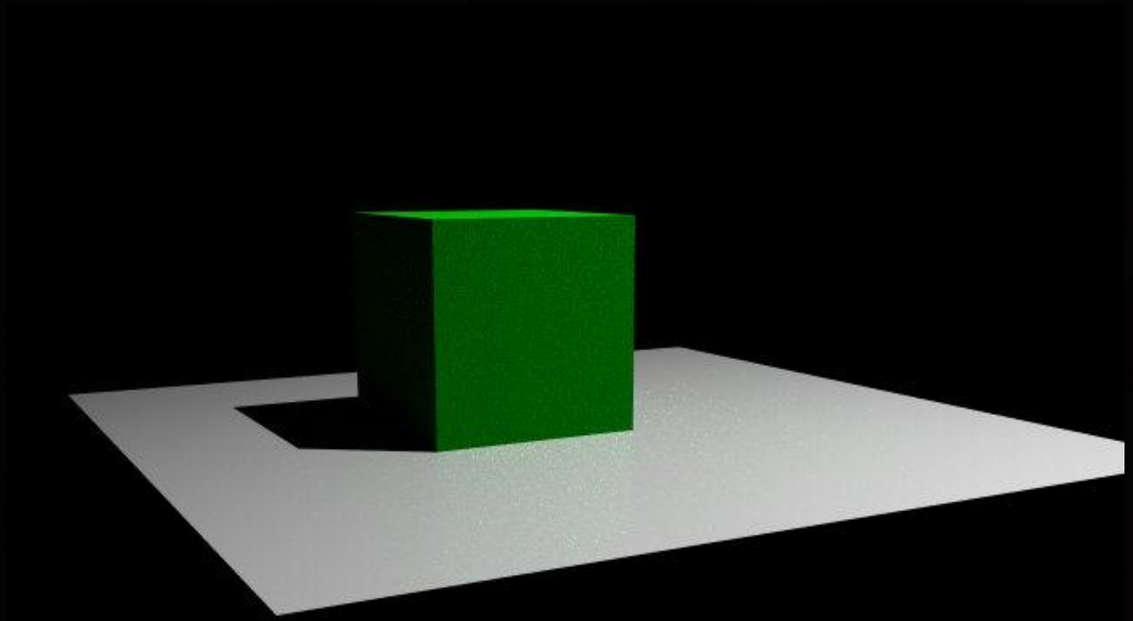
GET THE EXPORTER API

```
mi::base::Handle<mi::neuraylib::IExport_api> export_api(
        neuray->get_api_component<mi::neuraylib::IExport_api>());
```

WRITE THE IMAGE TO DISK

```
export_api->export_canvas("file:example_scene.png", canvas.get());
```

RESULT



ENVIRONMENT

CREATING THE HDR TEXTURE FROM FILE

```
string env_tex = mi::nrx::Scene_util::create_texture(trans, "C:\hdr\daytime.hdr");
```

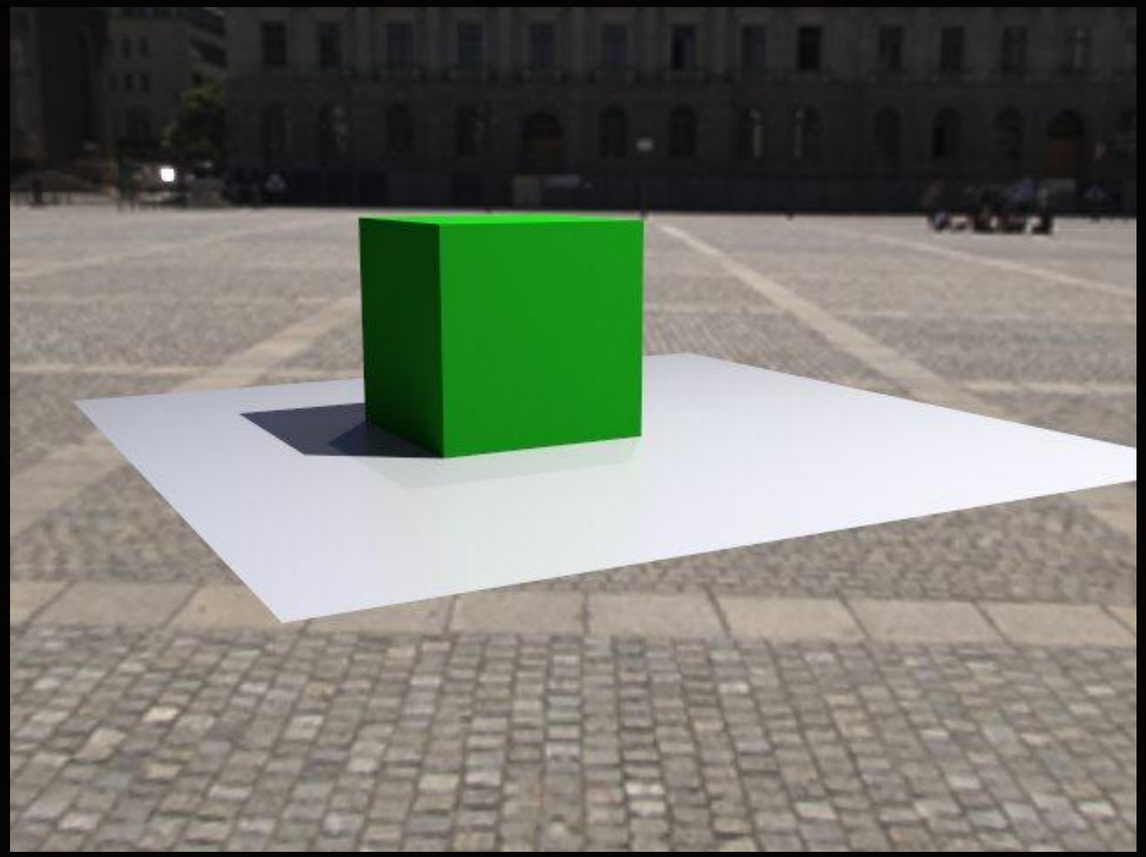
USING DEFAULT MDL ‘ENVIRONMENT SPHERICAL’ BASE FUNCTION

```
mi::nrx::Mdl env_shader(trans, neuray.factory(),
        mi::nrx::Mdl::get_base_function(trans, "environment_spherical").c_str(),
        "mat_env");
env_shader->set_value("texture", env_tex.c_str());
```

REFERENCING THE ENVIRONMENT SHADER FROM THE SCENE OPTIONS

```
mi::nrx::Attribute opt(trans, "options");
opt.create_attribute("environment_function", "Ref");
opt.set_value("environment_function", "mat_env");
```

RESULT



ENVIRONMENT SUN AND SKY

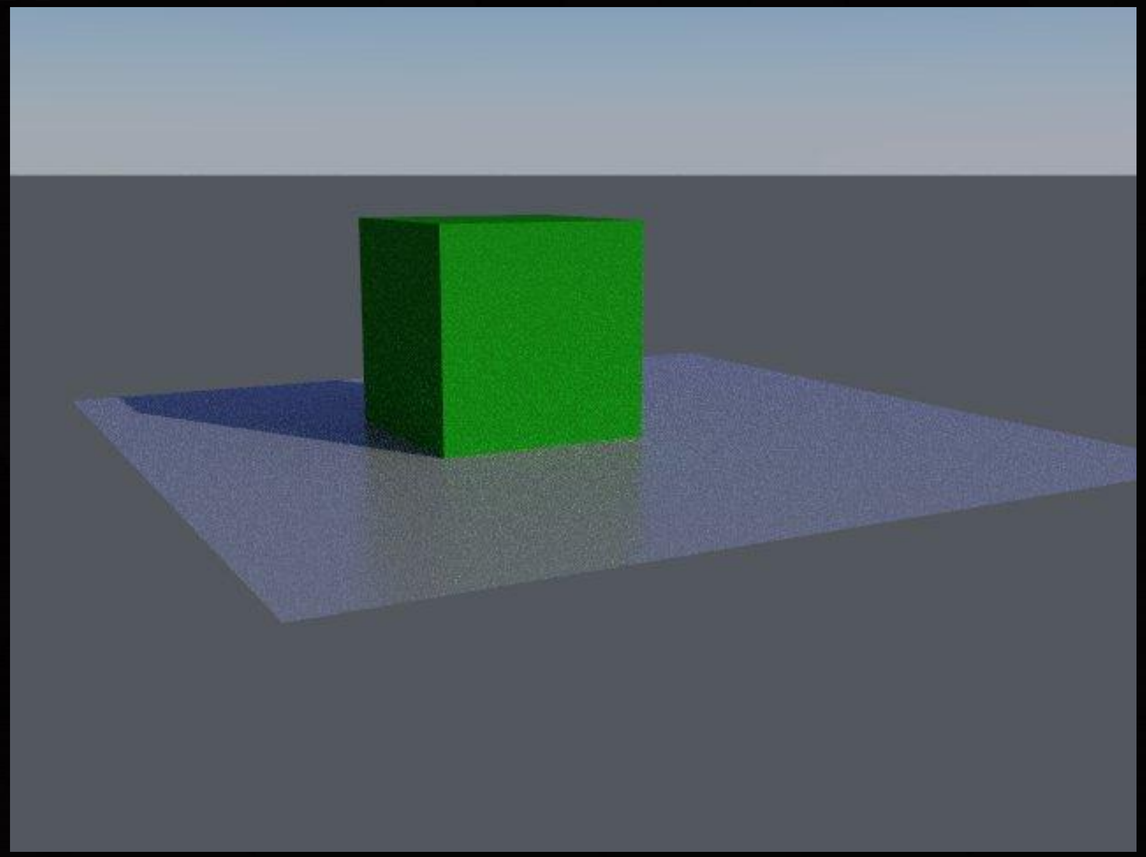
USING DEFAULT MDL ‘SUN AND SKY’ BASE FUNCTION

```
Mdl env_shader(transaction, neuray.factory(),
                    Mdl::get_base_function(trans, "sun_and_sky").c_str(), "mat_env");
```

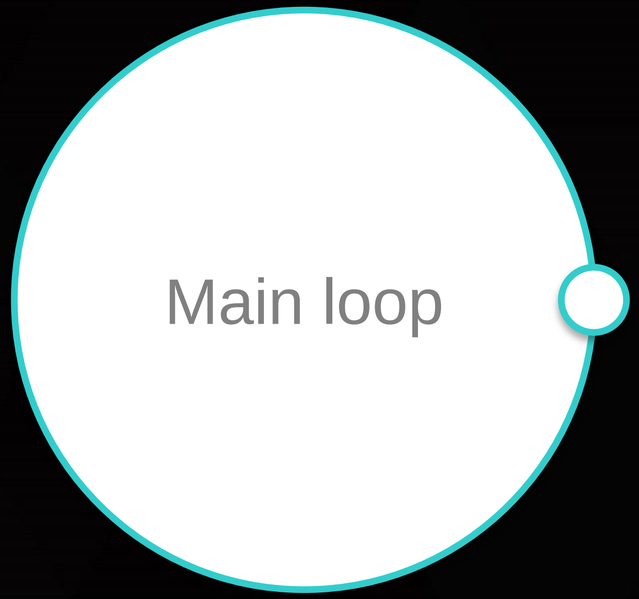
REFERENCING THE ENVIRONMENT SHADER TO THE SCENE OPTIONS

```
Attribute opt(transaction, "options");
opt.set_value("environment_function", "mat_env");
```

RESULT

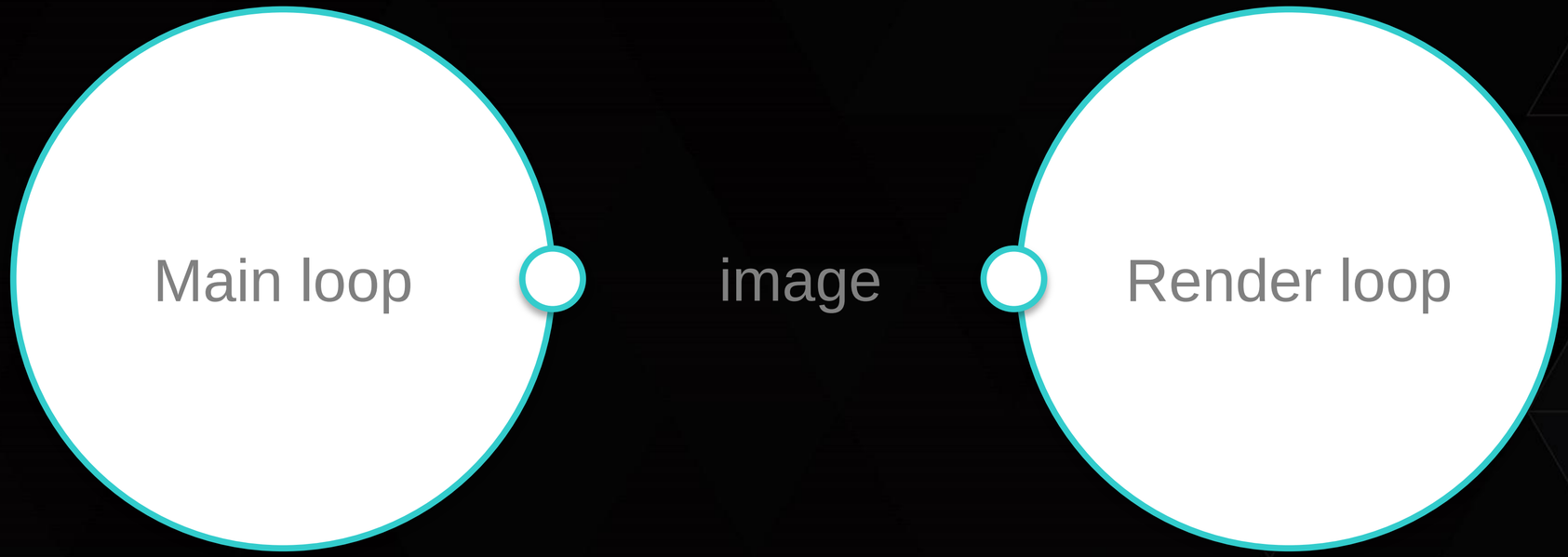


RENDERING

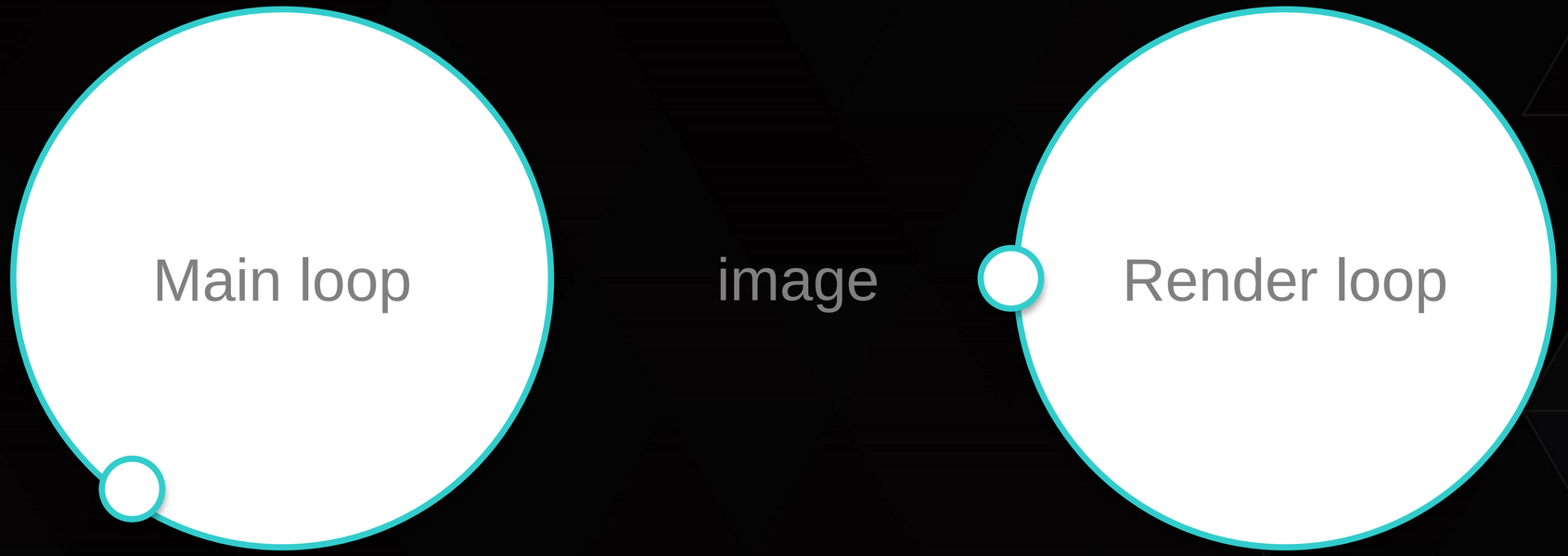


Render

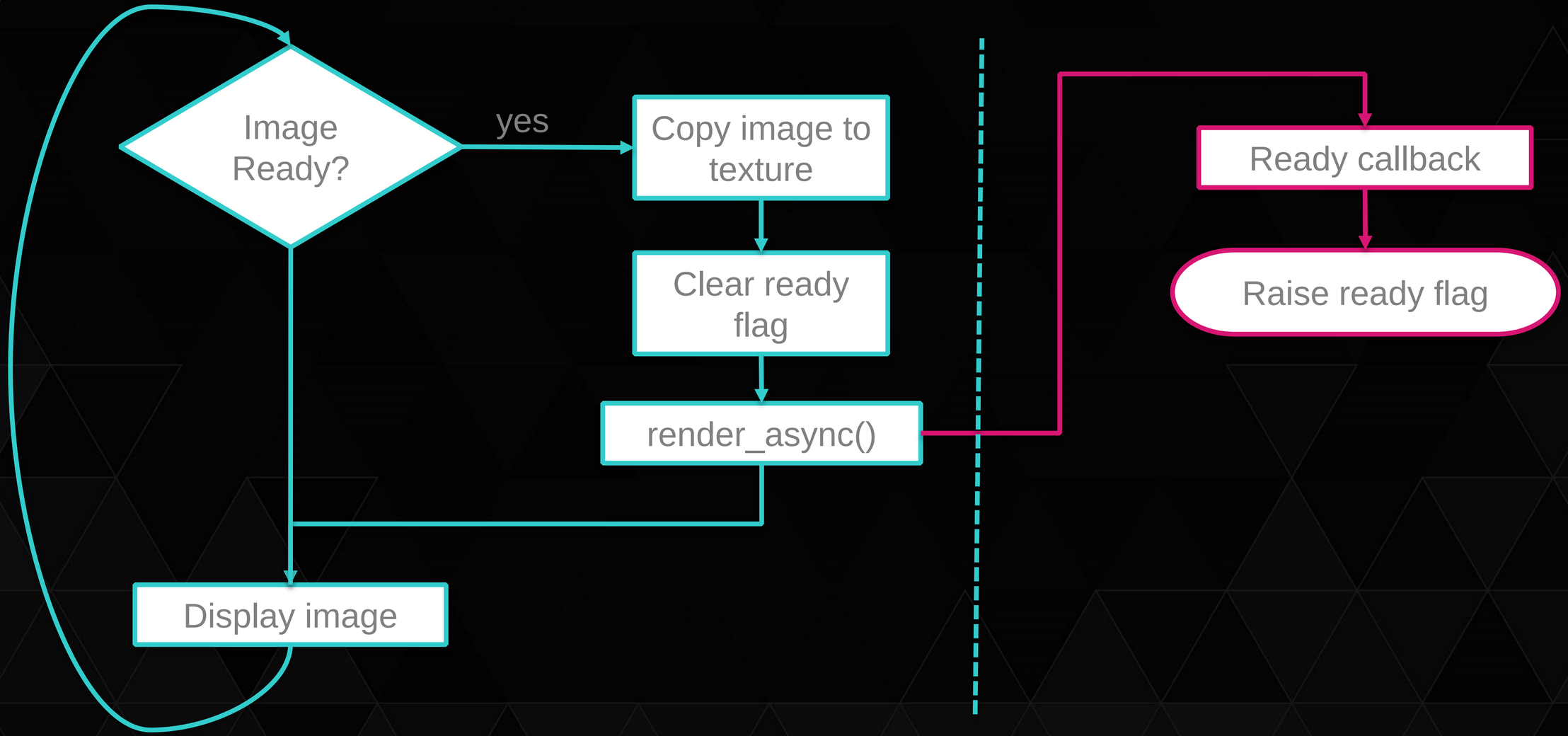
RENDERING



RENDERING



RENDERING INTERACTIVELY



RENDERING INTERACTIVELY

RENDERING LOOP

```
while (running)
{
    if (render_callback.ready())
    {
        copy_image();
        render_frame();
    }

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    Util_ogl::draw_texture_quad(ogl_tex_id);
}
```

COPY IMAGE

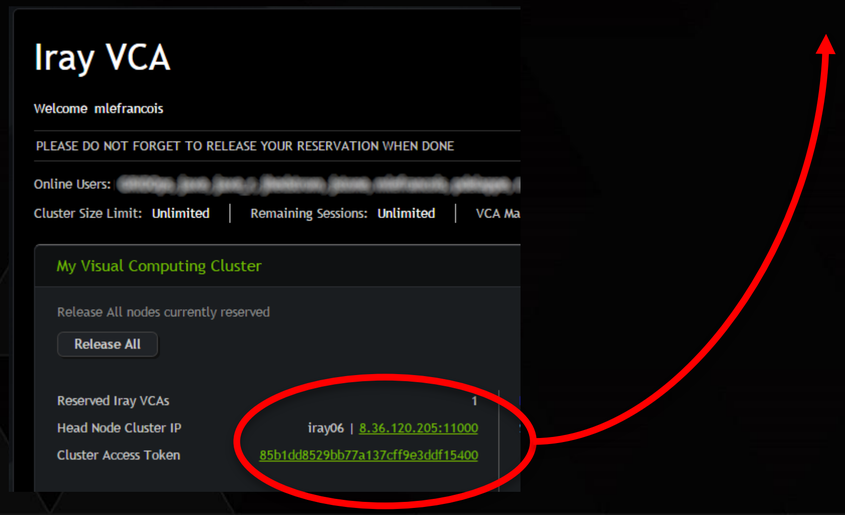
```
void copy_image()
{
    mi::base::Handle<mi::neuraylib::ICanvas> canvas(render_target->get_canvas(0));
    mi::base::Handle<mi::neuraylib::ITile> tile(canvas->get_tile(0, 0));
    glBindTexture(GL_TEXTURE_2D, ogl_tex_id);
    glTexImage2D(GL_TEXTURE_2D, 0,
                 Util_ogl::get_pixel_format_opengl(canvas->get_type()),
                 canvas->get_resolution_x(), canvas->get_resolution_y(), 0,
                 Util_ogl::get_pixel_format_opengl(canvas->get_type()),
                 Util_ogl::get_pixel_type_opengl(canvas->get_type()),
                 tile->get_data());
    glBindTexture(GL_TEXTURE_2D, 0);
}
```

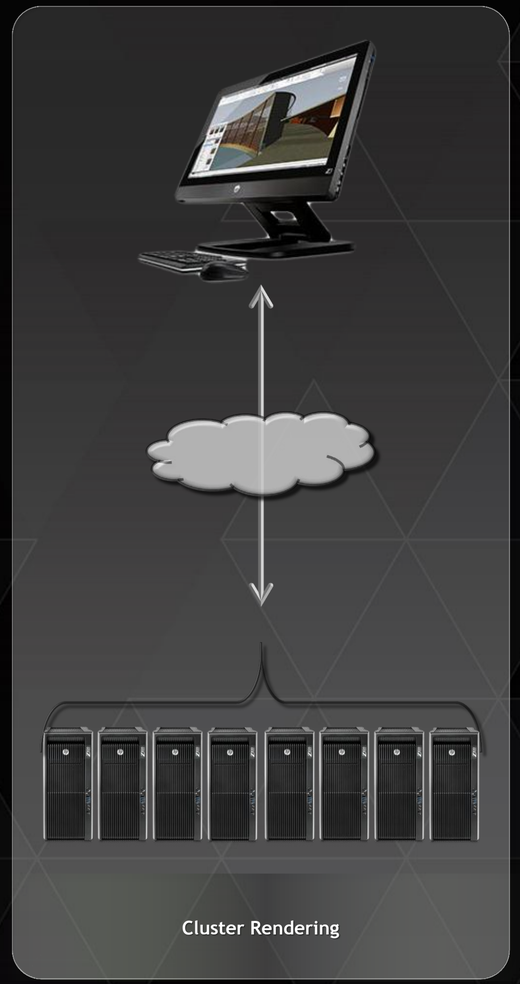
RENDER FRAME

```
void render_frame()
{
OPTIMIZATION: RE-CREATE A RENDERER TRANSACTION ONLY IF THE SOMETHING CHANGED IN THE SCENE
    if (neuray.trans_is_edited())
    {
        neuray.trans_edit_clear();
        if (render_transaction)
            render_transaction->commit();
        render_transaction = Handle<ITransaction>(neuray.scope()->create_transaction());
    }

CLEARING FLAG
    render_callback.start_rendering();

RENDERING IN SEPARATE THREAD
    render_context->render_async(render_transaction.get(), render_target,
                                                    &render_callback, &render_callback);
}
```

RENDERING USING VCAS

```
USE “NITRO_CLOUD” OR “IRAY_CLOUD”
  Handle<IRender_context> render_context(
       scene->create_render_context(transaction.get(), "iray_cloud"));

SET REMOTE SERVER ADDRESS AND SECURITY TOKEN
  Handle<mi::bridge::IIray_bridge_client> bridge_client(
    neuray->get_api_component<mi::bridge::IIray_bridge_client>());
  bridge_client->set_application_url("wss://8.36.120.205:11000");
  bridge_client->set_security_token("85b1dd8529bb77a137cff9e3ddf15400");
```





DEBUGGING

LOGGER

```
Handle<ILogging_configuration> logging_configuration(
                neuray->get_api_component<ILogging_configuration>());
logging_configuration->set_log_level(mi::base::MESSAGE_SEVERITY_ERROR);
logging_configuration->set_receiving_logger(my_logger);
```

HTTP ADMIN CONSOLE

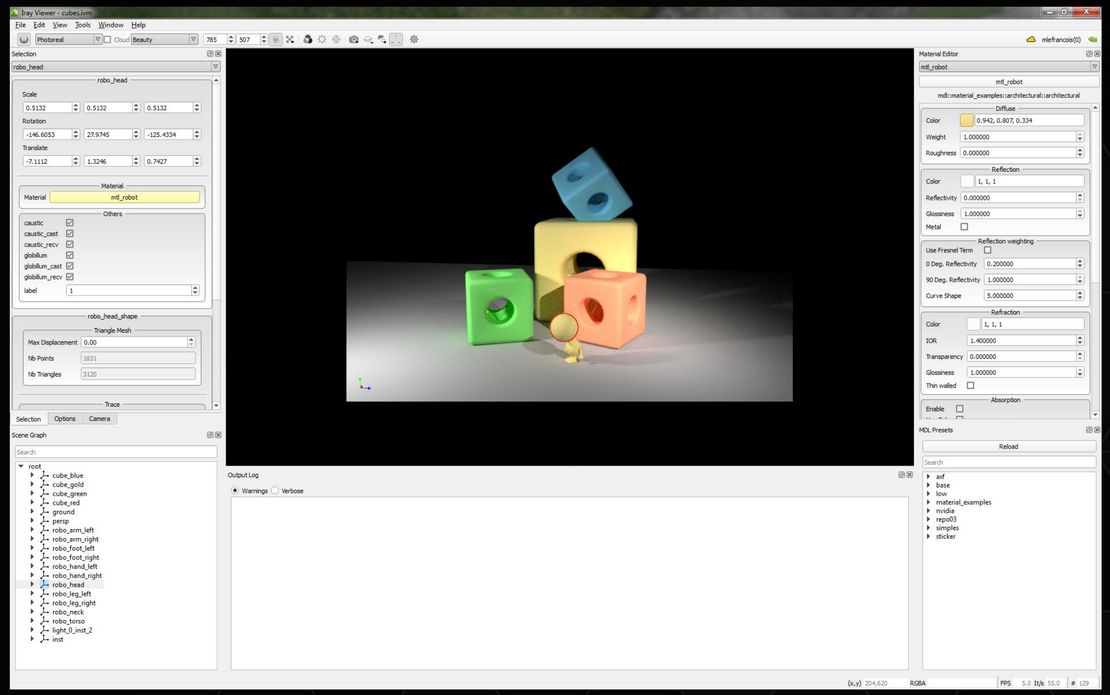
```
Handle<IGeneral_configuration>
    general_config(neuray->get_api_component<IGeneral_configuration>());
mi::Sint32 result = general_config->set_admin_http_address("0.0.0.0:10000");
```

EXPORT SCENE

```
mi::base::Handle< mi::neuraylib::IExport_api > export_api(
        neuray->get_api_component< mi::neuraylib::IExport_api >());
export_api->export_scene(transaction.get(),
        "file:my_scene.mi", "root_group", "camera", "options");
```

IRAY VIEWER

- Built using strictly the API
- Experiment features and options before implementing them
- Great to inspect scenes
- Measure performance
- Developing materials



RESOURCES

- HTML documentation: Full API documented
- Programmer’s Guide
- Code examples
- Forum: https://forum.nvidia-arc.com/forum.php

FURTHER INFORMATION

- Website
  - http://www.nvidia-arc.com/iray.html
- Email
  - irayintegration@nvidia.com

NVIDIA REGISTERED DEVELOPER PROGRAMS

- Everything you need to develop with NVIDIA products
- Membership is your first step in establishing a working relationship with NVIDIA Engineering
  - Exclusive access to pre-releases
  - Submit bugs and features requests
  - Stay informed about latest releases and training opportunities
  - Access to exclusive downloads
  - Exclusive activities and special offers
  - Interact with other developers in the NVIDIA Developer Forums

REGISTER FOR FREE AT: developer.nvidia.com

THANK YOU

JOIN THE CONVERSATION

#GTC15