[SD469655]

Autodesk Forge 平台在 M&E 行业中的应用

李成蹊
欧特克软件(中国)有限公司上海分公司

学习目标

- 介绍在 Forge 平台上处理 Maya 和 3ds Max 数据的各种方法
- 演示如何将 Forge 和 Shotgun 结合起来
- 探索如何在 M&E 工作流中使用 Forge Reality Capture
- 探讨 Forge 在 M&E 行业中的最佳实践以及一些小技巧

描述

最初 Forge 平台是针对 CAD 设计可视化及相关数据访问进行设计的。Forge Viewer(Large Model Viewer/LMV)的设计目的是在轻量平台展现大型 CAD 模型。它是用 Three.js 和 WebGL 打造的。但是 Viewer 本身的精度并不能始终满足 M&E 模型的需求。这就意味着不能使用 Forge 了嘛？当然不是！例如，尽管 Forge 没有提供 Maya 的直接支持， 我们仍然可以通过多种方式将数据上传到 Forge 中。 我们也可以将 Forge 与其他传统的 M&E 产品相结合（例如 Shotgun）。在本讲座中，我们将讨论如何将 Maya 的场景导入到 Forge 中以及相关的最佳实践。我们也会探索怎样将 Shotgun 的数据与 Forge 相结合。之后，我们也会对 3ds Max Design Automation 的应用进行探讨。最终我们会研究如何使用 Forge Reality Capture 服务将物件导入的您的场景中。

讲师

开发顾问

2014 年加入 Autodesk
自 2014 年起一直负责 Maya 的技术支持。了解 Maya API 和部分 Maya 的内部实现。
2019 年起加入 Forge 支持团队，目前主要负责点云和图形学相关的支持。

在 Forge 平台上处理 Maya 和 3ds Max 数据的各种方法

Maya

Forge 没有为 Maya 提供直接支持，但是我们可以通过导出 FBX 进行转换。导出 FBX 的时候需要注意同时导出材质和为 nurbs 曲线生成对应的 mesh。

SVF 提供对 3ds Max 物理材质的支持

3ds MAX 是欧特克主要的桌面三维设计软件之一。最近通过对 3ds Max Model Derivative 服务和 Forge Viewer 的更新，我们可以在网上查看带有 PBR 材质的 3ds Max 文件。

如果要使用这种方法，您需要在 3ds Max 场景中使用 PBR 材质并把上传到 Model Derivative 服务。您可以在下面的更新日志中了解这些功能：

- https://forge.autodesk.com/en/docs/model-derivative/v2/change_history/changelog/#release-date-2020-08-27
- https://forge.autodesk.com/en/docs/model-derivative/v2/change_history/changelog/#release-date-2020-06-02

图一是 3ds Max 中的 Viewport。图二是图一的文件 Forge Viewer 中的效果。图三是更新前使用 FBX 的效果(不支持 PBR)。

图一


24-105mm
ZOOM LENS
Forge Viewer showing 3ds Max format with PBR materials

图二


Forge Viewer showing 3ds Max scene exported
to FBX with materials embedded

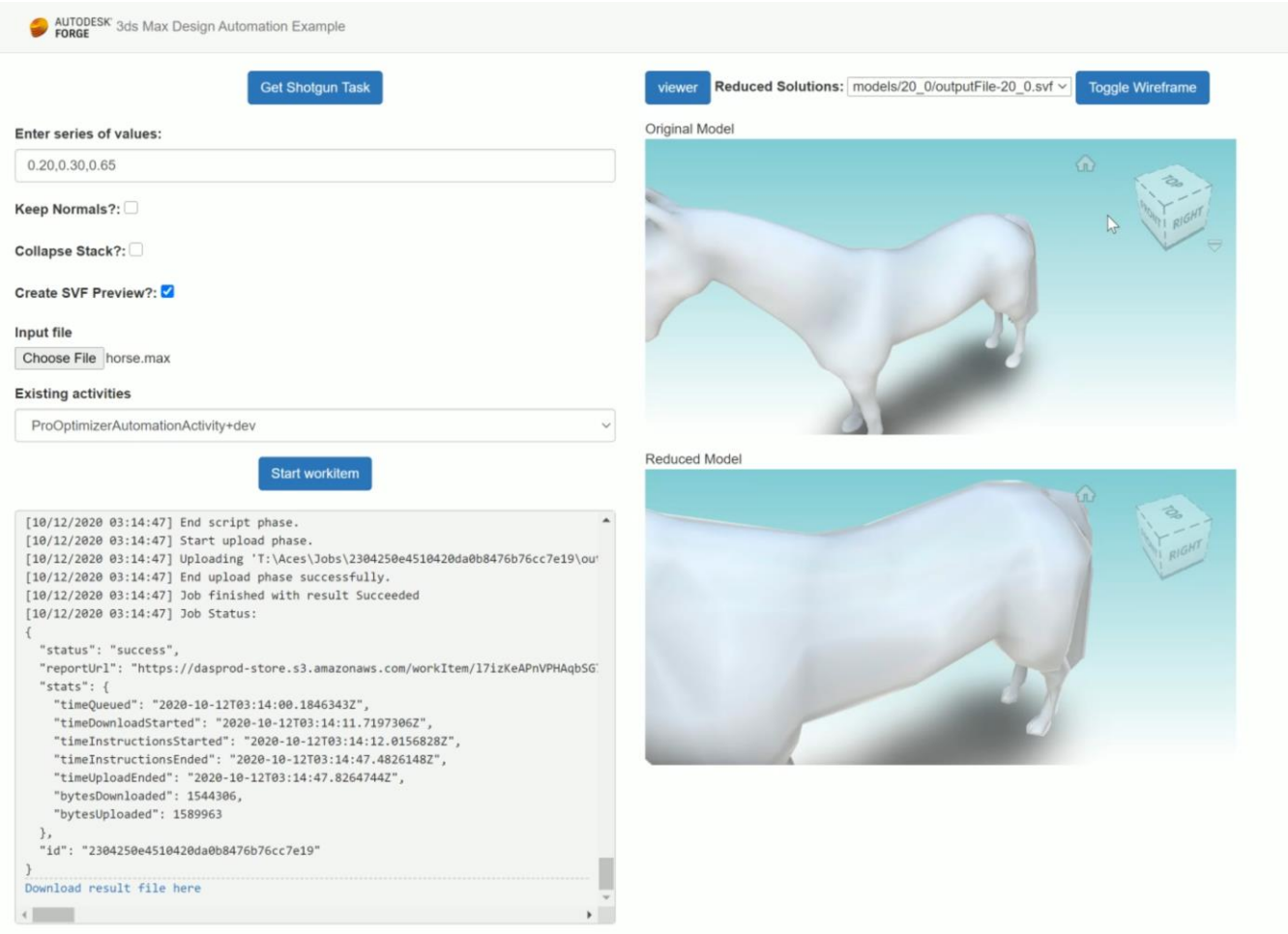
图 三

如何将 Forge 和 Shotgun 结合起来

Shotgun 是业内常用的项目和资源管理工具。这里我们将结合 Model Derivative、Design Automation 与 Reality Capture 服务来说明讲两者结合起来使用。

在 Forge 中使用 Shotgun 中的 Asset 设置

通过使用 3ds Max Design Automation，用户可以在云端完成以往只能通过桌面 3ds Max Batch 完成的操作。通过调用 Shotgun 的 API, 我们可以在使用 Design Automation 的时候调用相应的参数。我们将以一个样例来演示这个流程。



在这个样例中，我们使用了 3ds Max 中的 Pro Optimizer 插件，执行操作的参数则从 Shotgun 中获得。样例使用了 Forge .NET 的组件来开发，您也可以自己直接访问 Endpoint 来调用相应的服务。在使用 Design Automation 的时候，可以分为准备和执行两个步骤。

准备步骤中，我们可以选择要上传自己的插件(Appbundle)。这里我们需要提供使用设计自动化的引擎版本。

```
AppBundle appBundleSpec = new AppBundle()
{
    Package = appBundleName,
    Engine = engineName,
    Id = appBundleName,
    Description = string.Format("Description for {0}", appBundleName),

};
newAppVersion = await _designAutomation.CreateAppBundleAsync(appBundleSpec);
```

如果已经存在 Appbundle，我们可以为他更新一个版本

```
// create new version
AppBundle appBundleSpec = new AppBundle()
{
   Engine = engineName,
   Description = appBundleName
};
newAppVersion = await _designAutomation.CreateAppBundleVersionAsync(appBundleName, appBundleSpec);
if (newAppVersion == null) throw new Exception(message: "Cannot create new version");
```

在创建或者更新后，Forge 服务会返回一个地址供我们上传插件，插件是类似于 Autodesk exchange store 中 App 的格式。

在上传完 Appbundle 之后，我们开始创建 Activity。Activity 相当于自动化操作的定义，包含了使用的 Appbundle 和脚本。

```
Activity activitySpec = new Activity()
{
    Id = activityName,
    Appbundles = new List<string>() { string.Format("{0}.{1}+{2}", NickName, appBundleName, Alias) },
    CommandLine = new List<string>() { commandLine },
    Engine = engineName,
    Parameters = new Dictionary<string, Parameter>()
    {
        { "inputFile", new Parameter() { Description = "input file", LocalName = "$(inputFile)", Ondemand = false,
                                         Required = true, Verb = Verb.Get, Zip = false } },
        { "inputJson", new Parameter() { Description = "input json", LocalName = "params.json", Ondemand = false,
                                         Required = false, Verb = Verb.Get, Zip = false } },
        { "outputFile", new Parameter() { Description = "output file", LocalName = "output.zip" , Ondemand = false,
                                         Required = true, Verb = Verb.Put, Zip = false } }
    },
    Settings = new Dictionary<string, ISetting>()
    {
        { "script", new StringSetting(){ Value = engineAttributes.script } }
    }
};
```

上图的 Activity 中，通过设置 Verb 参数，我们创建了 input file 和 input json 两个输入和 output file 一个输出。在 script 设定中，我们提供了执行自动化时的脚本。

准备步骤完成后，我们可以通过 workitem 调用所创建的 Activity。

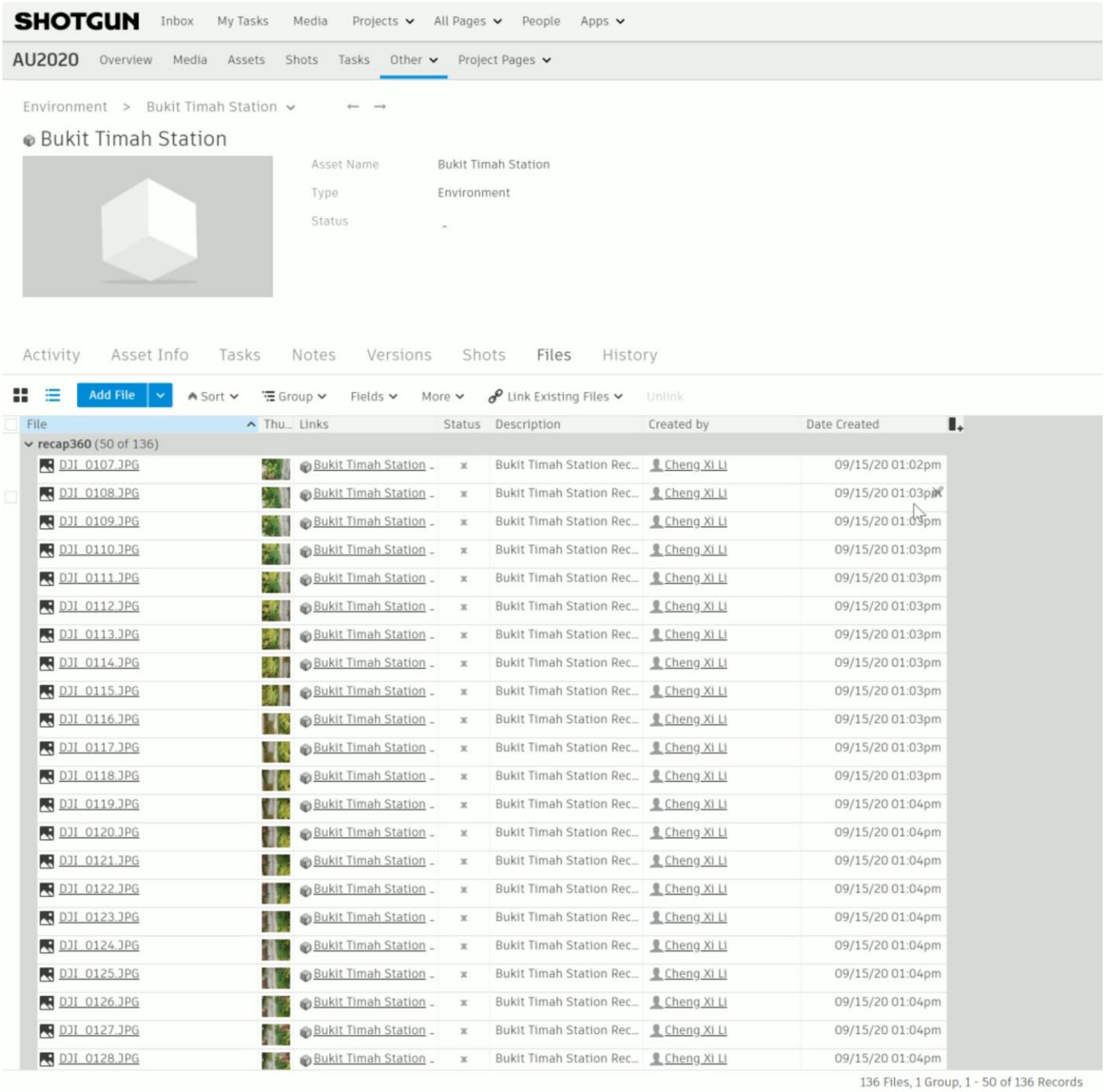
```
// prepare & submit workitem
string callbackUrl = string.Format("{0}/api/forge/callback/designautomation?id={1}&outputFileName={2}",
    OAuthController.GetAppSetting("FORGE_WEBHOOK_URL"), browerConnectionId, outputFileNameOSS);
WorkItem workItemSpec = new WorkItem()
{
    ActivityId = activityName,
    Arguments = new Dictionary<string, IArgument>()
    {
        { "inputFile", inputFileArgument },
        { "inputJson",  inputJsonArgument },
        { "outputFile", outputFileArgument },
        { "onComplete", new XrefTreeArgument { Verb = Verb.Post, Url = callbackUrl } }
    }
};
WorkItemStatus workItemStatus = await _designAutomation.CreateWorkItemsAsync(workItemSpec);
```

填写 Activity 对应的字段后，我们就可以提交了。样例中，我们使用了 Callback 来接受服务器端完成任务的提醒。当自动化处理完成，在接收到服务器端的通知后，我们就可以下载文件了。

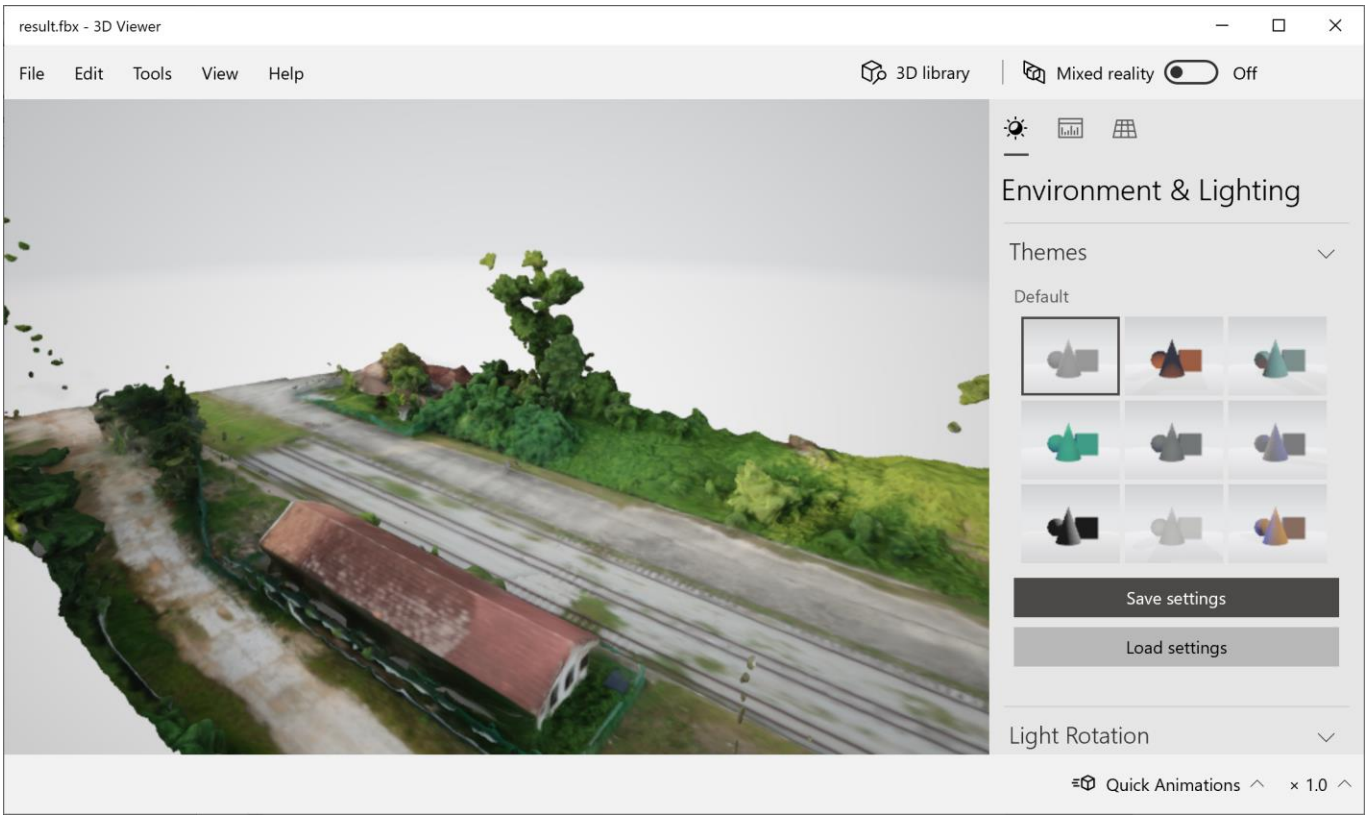
在 Shotgun 中调用 Forge Reality Capture API

在创作 Asset 的时候，我们会添加一些参考素材。如果参考素材拍摄符合照片建模的标准，我们可以直接使用 Reality Capture API(简称 Recap)生成一个简单的模型。这个样例介绍了如何配置 Shotgun 以及如何使用 Reality Capture 来生成参考模型。

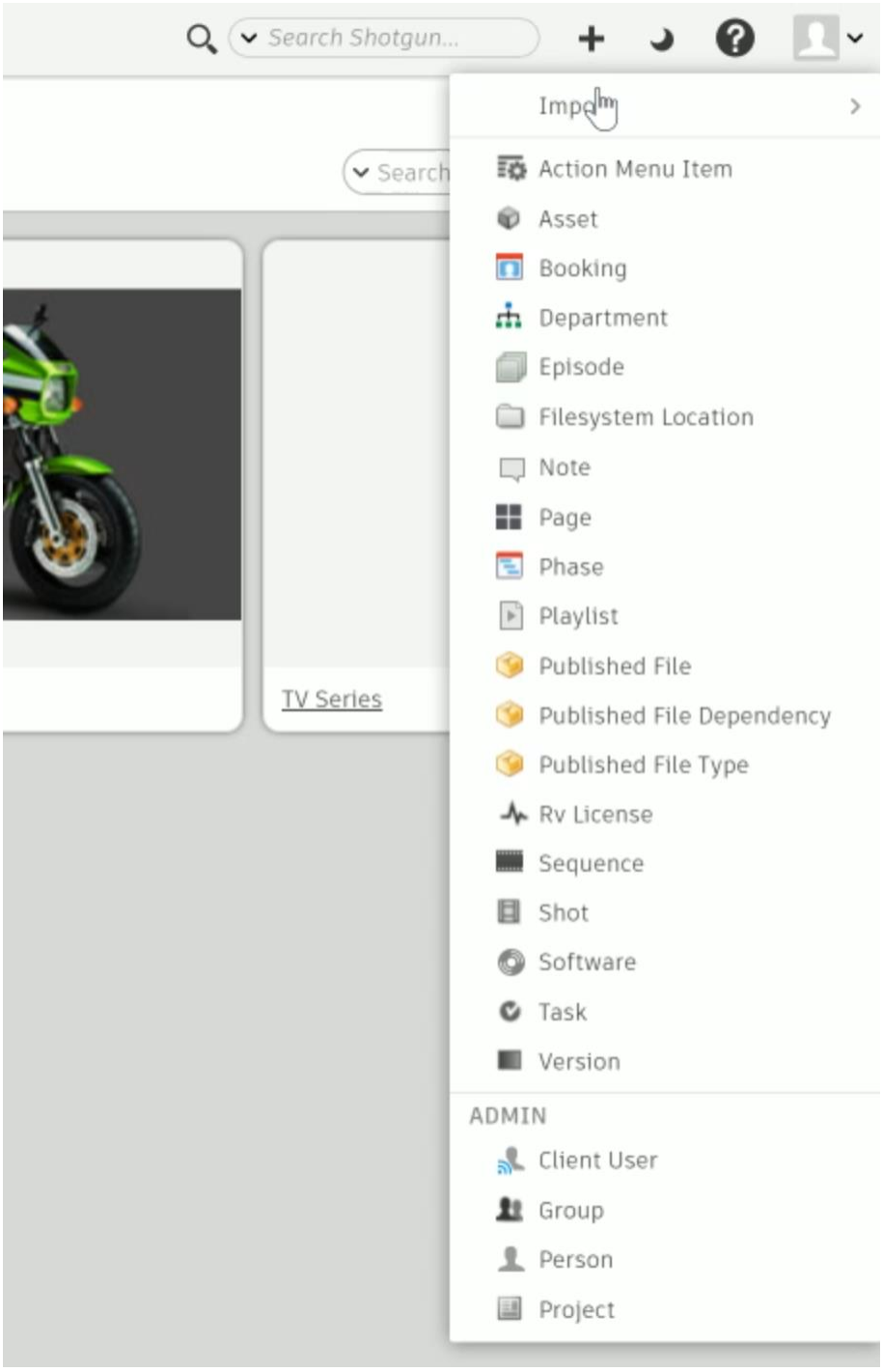
在样例中，我可以将 Asset 中的图片直接提交到 Reality Capture 服务中。



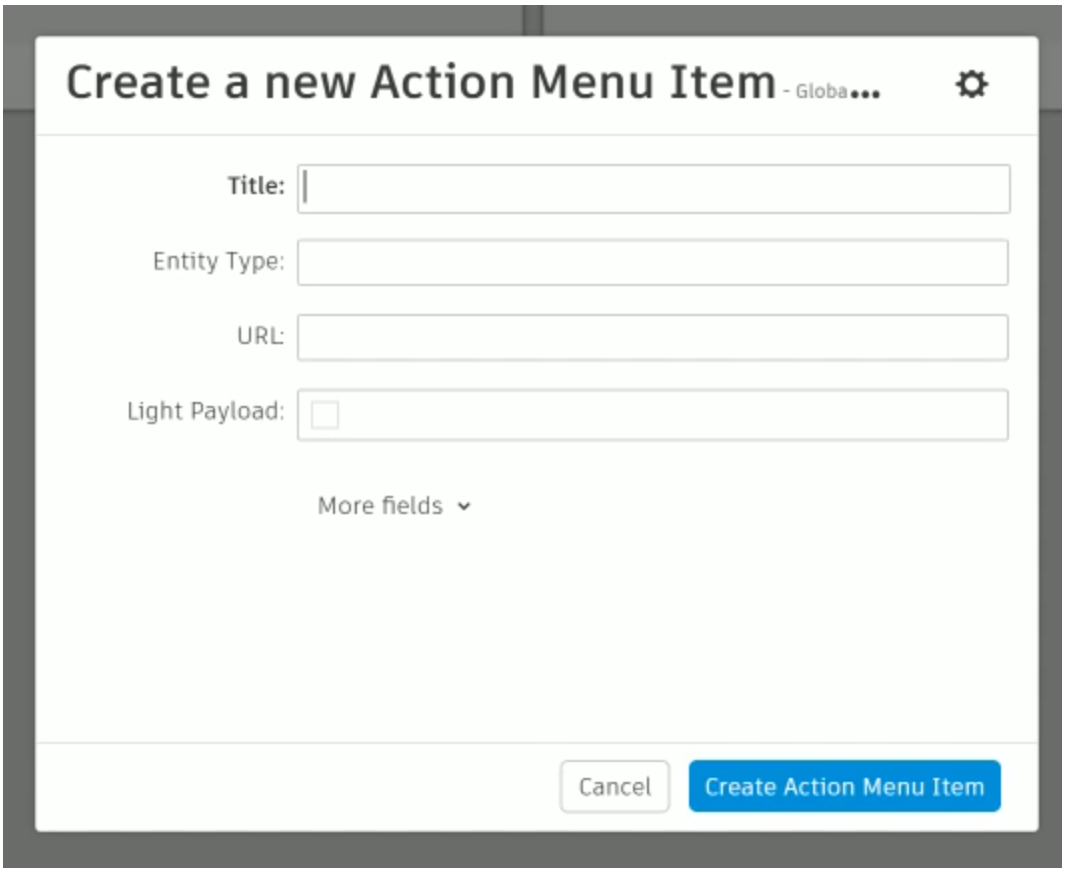
最后得到如下的场景模型



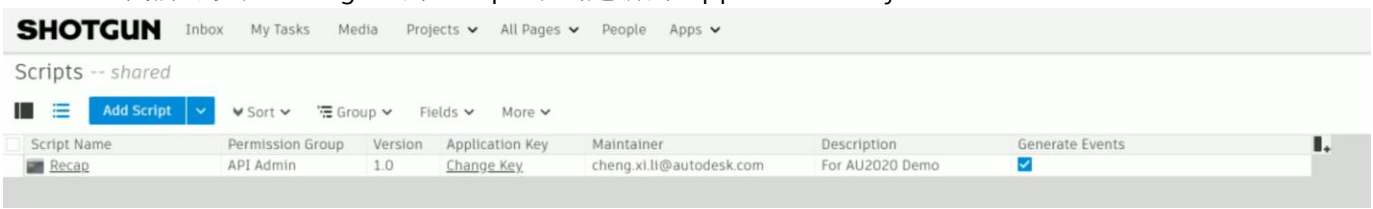
为了实现这个服务，我们首先要在 Shotgun 中创建 AMI。



在创建 AMI 的时候我们可以指定名称，服务地址。



为了使 Recap 的 Webservice 可以访问到 Shotgun，我们除了使用用户名密码外，还可以使用 Secret。我们可以在 Shotgun 的 Script 中创建新的 Application Key。



在配置好后，我们就可以通过 Webservice 来访问 Shotgun 了。需要注意的是，在服务器端认证的时候，Script name 需要与 Application Key 一致才可以登录。

通过 AMI 和 Shotgun 的 Webservice，我们可以得到用户所指定的文件的直接下载地址。

生成一个 Recap 场景的时候，我们要先创建一个 Photo Scene。这个时候我们需要指定他的照片拍摄手法，坐标信息以及输出格式。例如：

```
def CreatePhotoScene(name, scene_type='aerial', gps_type='regular', callback=''):
    global RecapPhotoSceneLocation
    callback_base = os.getenv('RECAP_CALLBACK_URL')
    if callback_base is None:
        print("Error: Missing RECAP_CALLBACK_URL in your environment")
        return None
    data = {
        'scenename': name,
        'format': 'rcm,fbx',
        'scenetype': scene_type,
        'gpstype': gps_type,
        'callback': f'{callback_base}/{callback}'
    }
```

我们创建了一个场景，输出格式包括 Recap 格式(RCM)和 FBX。默认为航拍图片，坐标系统为 Regular。在执行完毕后，Recap 服务器就会调用 Callback 地址来通知服务器任务完成。

接下来是上传照片，需要注意的是，上传的时候往往因为文件较多较大，速度会较慢。推荐先返回给用户作业创建成功的提示，再进行异步上传。

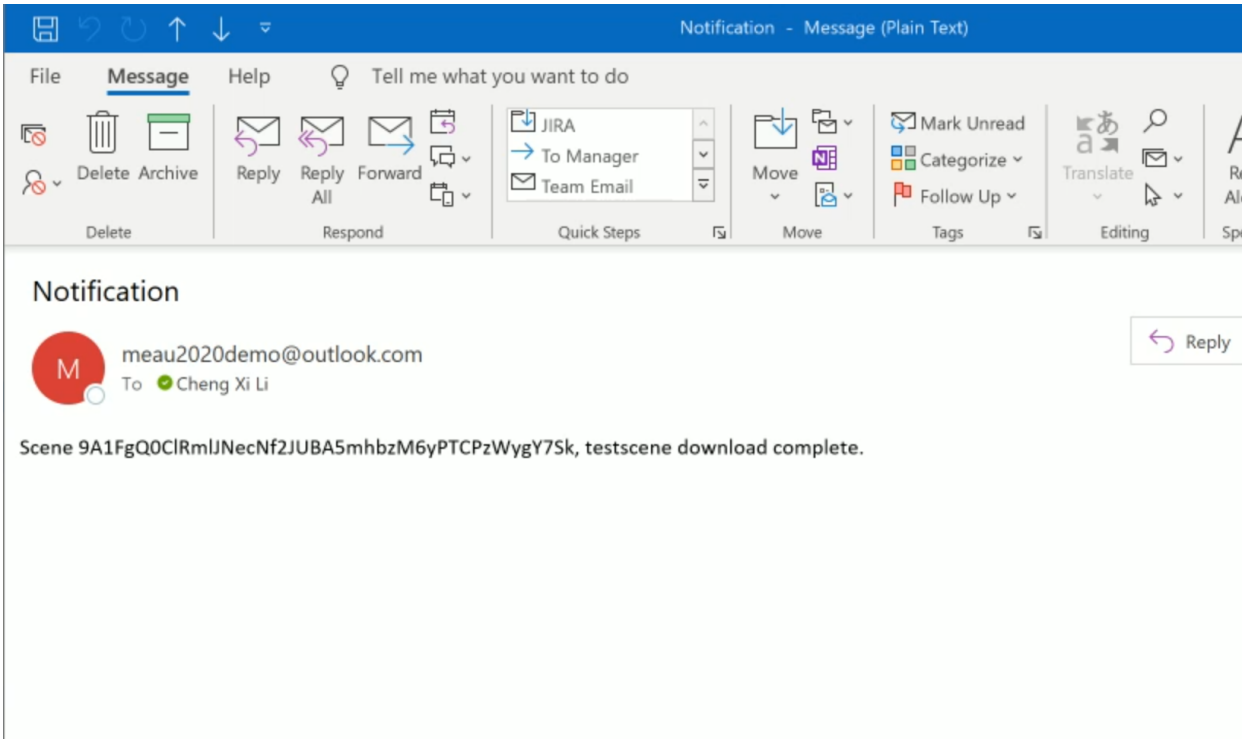
最后在上传完文件后，再 Post Scene name 给 Recap 服务器的 photoscene endpoint 即可。

```
def PostScene(scene_name):
    global RecapPhotoSceneLocation
    uri = f'{RecapPhotoSceneLocation}/{scene_name}'
    headers = getUrlFormHeader()
    r = requests.post(uri, headers=headers)
    return r.content
```

当收到 Callback 时，我们就可以下载对应的场景了。如果希望下载 RCM 以外的格式，我们需要提供 format 参数。样例中，我们需要 FBX 格式，所以应该加上?format=fbx

```
def DownloadPhotoScene(name, filename, format='fbx'):
    global RecapPhotoSceneLocation
    local_filename = f'{RecapFileStorageLocation}\\{filename}.{name
    url = f'{RecapPhotoSceneLocation}/{name}?format={format}'
    response = requests.get(url, headers=getUrlFormHeader())
```

在下载完成后，样例中采用了 Email 的方法来通知用户，这也是 Shotgun 中的常见做法之一。



演讲中的两个样例可以在 Github 上下载到，链接如下：

- https://github.com/kevinvandecar/design.automation.3dsmax-csharp-meshoptimizer/tree/viewer+shotgun-support
- https://github.com/iamsleepy/Shotgun.Recap