Introduction

Cosm3D is a library that extends Cosmos VMwareSVGAII capabilities with 3D rendering.

In this tutorial series we will learn how to download Cosmos create a project, install and use Cosm3D and how to setup VMware to work with Cosm3D

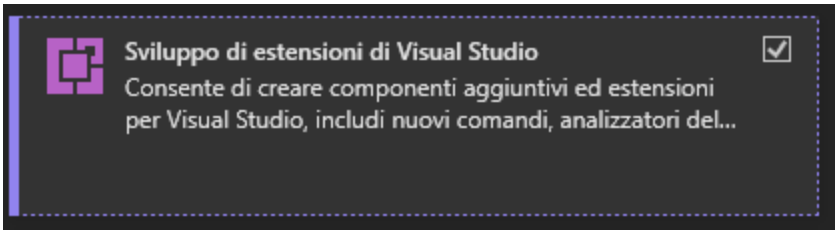
Installing Cosmos

Prerequisites

- Windows10/11 or Linux
- .Net6
- vcpp2010
- VisualStudio2022 (Windows only)

Installing the workloads (you do not need VS2022 if you are on Linux)

Cosmos requires a workload for Visual Studio to work. To install it, select the Extensions Development workload in the Visual Studio installer before installing Visual Studio 2022, or if you already have it installed, you can modify the install and add the workload.



Downloading the devkit

If you have Git installed, run this command in an empty folder:

```
git clone https://github.com/CosmosOS/Cosmos.git
```

Else, you can go to the CosmosOS repo and clone it manually. Remember to put it in an empty folder.

After cloning the repo, you need to rename the Cosmos-master folder to Cosmos.

Building the devkit

Windows

Run the install-VS2022.bat file with administrator privileges and make sure to have Visual Studio closed.

Linux

In the makefile change line 19 from:

```
BUILDMODE=Release
```

to:

```
BUILDMODE=Debug
```

Open a terminal in the Cosmos folder and run the make command. You might want to also install the Cosmos template. To install it, run:

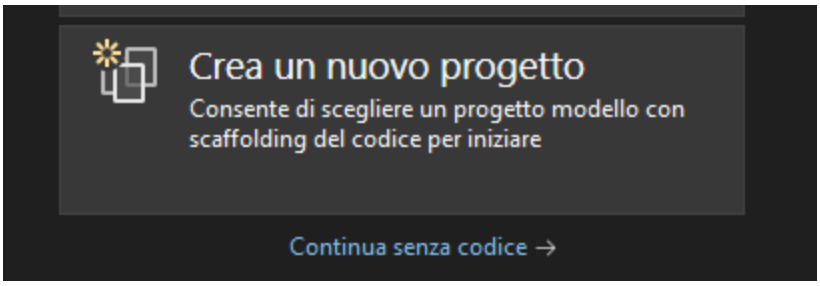
```
dotnet new install source/templates/csharp
```

Optionally, you can install CosmosCLI for better project management on Linux.

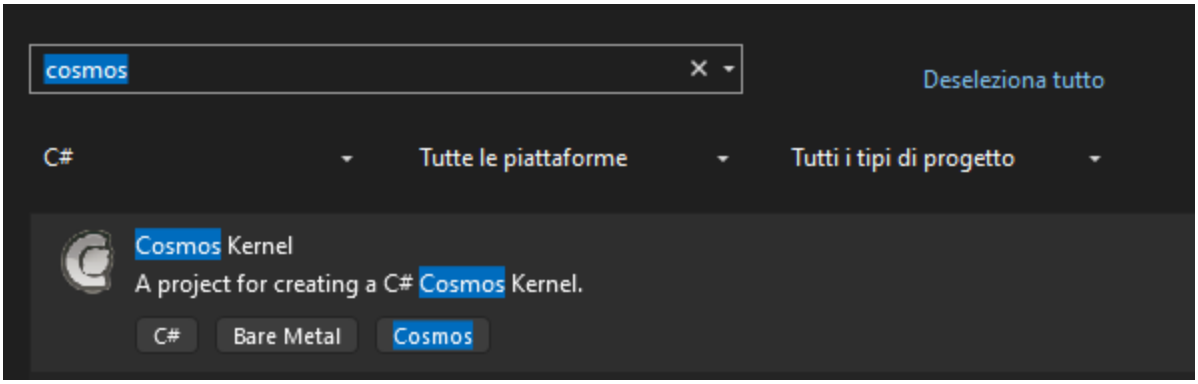
Creating a project

Windows

To create a Cosmos project, open Visual stuio 2022 and press **Create a new project**



Then Search and select "Cosmos Kernel".

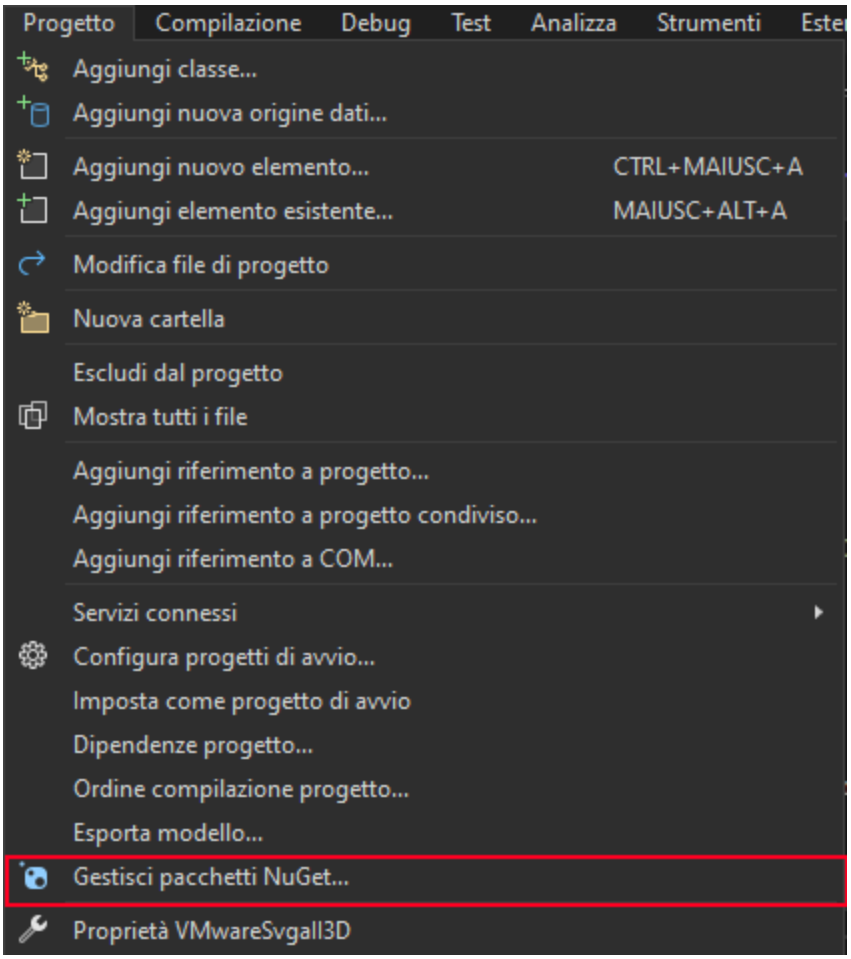


Click **next**, give the project a name (int this tutorial series we will use "LearnCosm3D") and click **create**

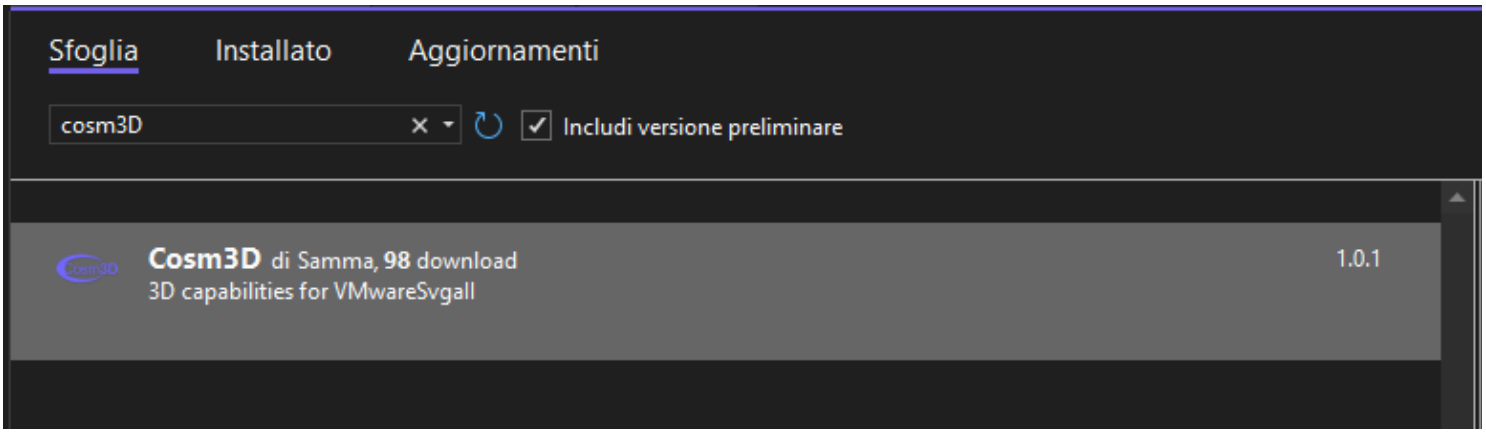
Linux

Installing Cosm3D

To install Cosm3D click on **project -> manage nuget packages**



Then search for "Cosm3D", select the first resul and press install



Creating a context

Now that Cosmos and Cosm3D are setted up we can start coding.

The first thing to do is creating a 3D VMwareSVGAII canvas.

```
namespace LearnCosm3D
{
    public class Kernel : Sys.Kernel
    {

        SVGAII3DCanvas canv;

        protected override void BeforeRun()
        {
            canv = new SVGAII3DCanvas(new Mode(1280,720,ColorDepth.ColorDepth32));
        }

        protected override void Run()
        {
        }
    }
}
```

the constructor for `SVGAII3DCanvas` takes one arument, that argument is a `Mode`.

a `Mode` is the resolution that we want to use. here we are creating a mode that is 1280x720 pixels in size and that has 32 bits of color depth.

Next up is defining a Context.

```
public class Kernel : Sys.Kernel
{

    SVGAII3DCanvas canv;
    uint context;

    protected override void BeforeRun()
    {
        canv = new SVGAII3DCanvas(new Mode(1280,720,ColorDepth.ColorDepth32));
        context = canv.driver.DefineContext();
    }

    protected override void Run()
```

```
        {
        }
}
```

A context is an id that is used to distunguish between different 3D environments

Creating a context

A surface is an image that the gpu has access to, and that we can render to.

We will need two surfaces, one for Color and one for Depth.

```
SVGA3dSurfaceImageId Color, Depth;

protected override void BeforeRun()
{
    // ...

    Color = canv.driver.DefineSurface(1280, 720, SVGA3dSurfaceFormat.SVGA3D_X8R8G8B8);
    Depth = canv.driver.DefineSurface(1280, 720, SVGA3dSurfaceFormat.SVGA3D_Z_D16);
}
```

A `SVGA3dSurfaceImageId` is the object type that contains the id of our surfaces, similarly to the canvas, our surfaces need a width and height and a format, the format in the case of our `Color` surface is `SVGA3D_X8R8G8B8`, indicating a 24 bit RGB color component with an unused 8 bit component. Our `Depth` surface is of format `SVGA3D_Z_D16`, indicating a 16 bit Depth component.

To render to the surfaces that we just made, we first have to set them as render targets.

```
SVGA3dSurfaceImageId Color, Depth;

protected override void BeforeRun()
{
    // ...

    Color = canv.driver.DefineSurface(1280, 720, SVGA3dSurfaceFormat.SVGA3D_X8R8G8B8);
    Depth = canv.driver.DefineSurface(1280, 720, SVGA3dSurfaceFormat.SVGA3D_Z_D16);

    canv.driver.SetRenderTarget(context, SVGA3dRenderTargetType.Color, Color);
    canv.driver.SetRenderTarget(context, SVGA3dRenderTargetType.Depth, Depth);
}
```