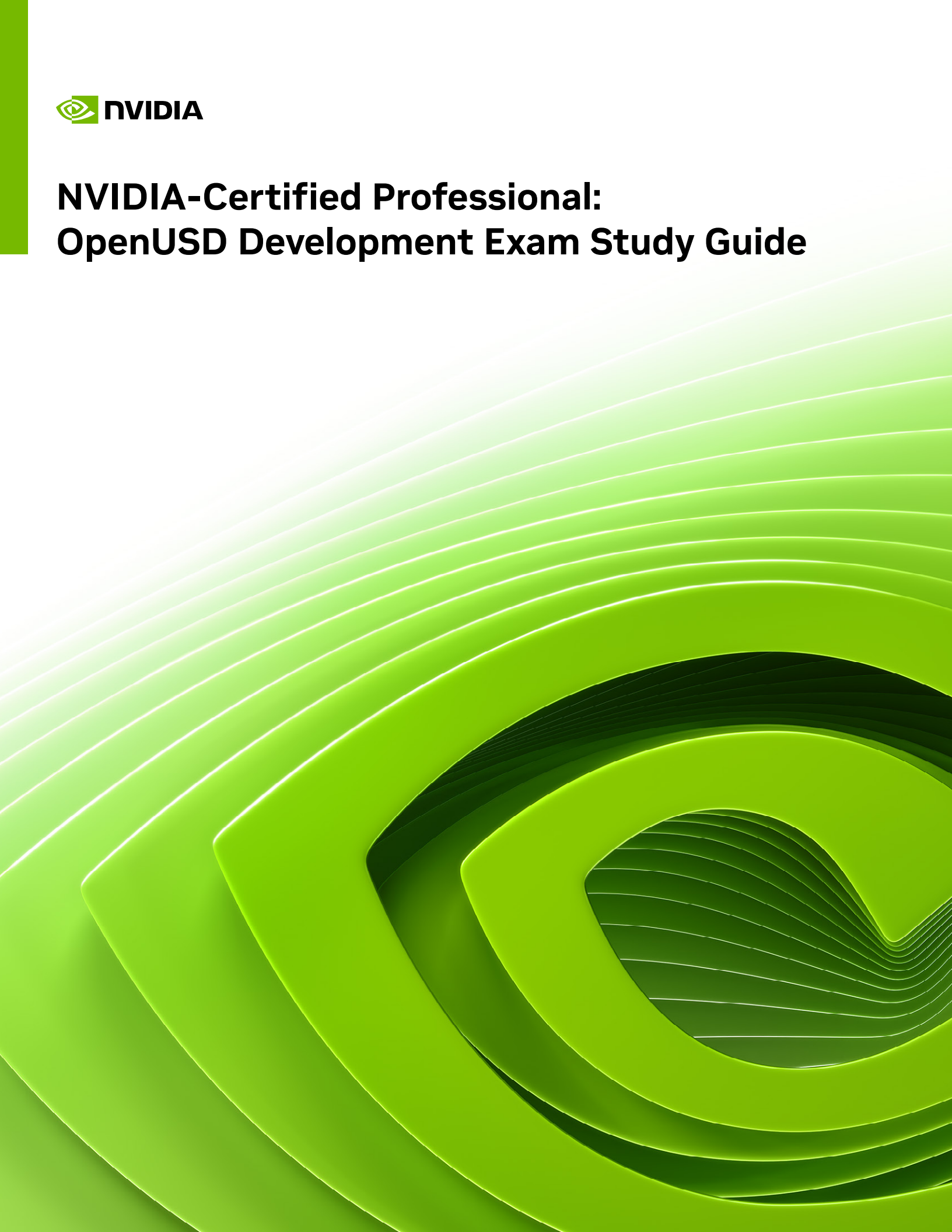
NVIDIA

NVIDIA-Certified Professional: OpenUSD Development Exam Study Guide



NVIDIA-Certified Professional: OpenUSD Development Exam Study Guide

Contents

Introduction

This study guide provides an overview of each topic covered on the NVIDIA OpenUSD Development certification exam, recommended training, and suggested reading to prepare for the exam.

Job Description

The OpenUSD developer is a technical expert who specializes in building, maintaining, and optimizing 3D content creation pipelines using OpenUSD. These engineers work at the intersection of 3D content development and software engineering, focusing on integrating complex tools and workflows to ensure seamless collaboration across teams such as artists, developers, and technical directors.

They are typically employed by companies in industries like animation, visual effects, gaming, manufacturing, AECO, and virtual production. They possess a deep understanding of 3D data structures, rendering systems, and the OpenUSD framework to enable efficient asset interchange, collaboration, and scene management.

Key Responsibilities

- Develop and maintain OpenUSD-based 3D content creation pipelines.
- Integrate OpenUSD with existing tools and optimize 3D asset performance.
- Collaborate with domain experts to automate tasks and solve pipeline issues.
- Document processes and train team members on OpenUSD pipeline use.
- Stay current on OpenUSD advancements to drive innovation and scalability.

Recommended Qualifications and Experience

- Expertise in OpenUSD framework and Python/C++ programming.
- 3D content creation, problem-solving, and version control skills.
- Cross-team communication and understanding of rendering systems.
- Performance optimization and continuous learning mindset.

Certification Topics and References

Composition: Exam Weight 23%

Authoring, design with, or debugging composition arcs. A developer needs to know all of the composition arcs, how they work, and when and when it is appropriate to use each. The developer needs to be able to debug complex LIVRPS scenarios.

1.1	Change the strength of an opinion.
1.2	Choose an appropriate instancing style for data at different scales.
1.3	Compare between referencing, payloads, and sublayers, and identify their use cases.
1.4	Create a scene with multiple elements referencing the same animation at different time-offset.
1.5	Design potential scene layering strategies to facilitate multi-user workflows.
1.6	Explain LIVERPS in simple terms.
1.7	Identify situations where variants are/are not appropriate for structuring assets.
1.8	Identify why opinions from one layer do not take effect in a composed stage.
1.9	Prepare an internal asset to be delivered to external parties.
1.10	Remove properties from instanced component prims in an assembly stage.
1.11	Split a monolithic asset to model multiple collaborative workstreams.

NVIDIA Courses and Suggested Readings

Course Reference

- Learn OpenUSD: Creating Composition Arcs

Tutorials

- Referencing Layers
- Authoring Variants

Glossary

- LIVERPS
- Sublayers
- Variant Set
- Reference
- Payload
- Timesample
- Value Clips
- Layer Offset
- Edit Target
- Change Processing
- Flatten

API Docs

- Scenegraph Instancing
- Basic Datatypes for Scene Description Provided by Sdf
- Edit Target - Detailed Description
- UsdStage::Flatten()
- UsdReference - Expressing References Without Prim Paths

Additional Resources

- USD Composition - SIGGRAPH 2019
- USD Book - Default Prim
- OpenUSD Code Samples - Add a Payload

Content Aggregation: Exam Weight 10%

Create modular, reusable components, leverage instancing (native and point) to optimize a scene, and apply different strategies for overriding an instanced asset for efficient, optimized, and collaborative aggregation of assets (models) to build large scenes.

2.1 Add a new prototype to a pointInstancer.
2.2 Edit the color of an instanceProxy mesh without changing its instancing status.
2.3 Hide instances of a pointInstancer efficiently.
2.4 Implement and manage USD instances for efficient asset reuse in large-scale scenes.
2.5 Remove properties from instanced component prims in an assembly stage.

NVIDIA Courses and Suggested Readings

Course Reference

- Learn OpenUSD: Understanding Model Kinds
- Learn OpenUSD: Asset Structure Principles and Content Aggregation
- Learn OpenUSD: Asset Modularity and Instancing

API Docs

- Scenegraph Instancing
- Kind: Extensible Categorization - The Core Kind Hierarchy
- Important Properties of Scene Description - Model Hierarchy: Meaning and Purpose
- UsdShadeNodeGraph - Detailed Description
- UsdGeomPointInstancer - Detailed Description

Customizing USD: Exam Weight 6%

Understand USD plugin development to extend USD's functionality, including the creation of custom schemas, file format plugins, custom model kinds, and variant fallback selections.
3.1 Build a USD plug-in against a given version of USD.
3.2 Build USD from scratch with custom dependencies.
3.3 Create custom model kinds when appropriate.
3.4 Create custom schemas for proprietary data models.
3.5 Integrate a custom resolver to manage asset paths dynamically.
3.6 Use USD schemas to support nonstandard attributes/structures during import/export.
3.7 Write a SceneIndex plug-in to generate renderable geometry directly as Hydra prims.
3.8 Write an AssetResolver that generates in-memory renderable primitives.

NVIDIA Courses and Suggested Readings

Course Reference

> Learn OpenUSD: Understanding Model Kinds

Tutorials

> Generating New Schema Classes

Glossary

> Model Hierarchy

Whitepapers and Blogs

> What Are OpenUSD Schemas?

Videos

> Universal Scene Description (OpenUSD): Custom Schemas
> USD in Production

Technical Reference

> USD FAQ - Isn't USD Just Another File Format?

API Docs

> Creating a File Format Plug-in
> Creating New Schema Classes With usdGenSchema
> Kind: Extensible Categorization
> PlugRegistry - Registering Plug-ins
> Ar: Asset Resolution

Data Exchange: Exam Weight 15%

Creating conceptual data mapping documents, custom importers, exports, and scripts for interchange of data with OpenUSD.
4.1 Convert OpenUSD assets to common 3D formats (such as glTF), and ensure fidelity.
4.2 Document conceptual mappings between USD and another data model (e.g., materialX).
4.3 Explain the trade-offs between USDC and USDA formats (performance, readability, archival, etc.).
4.4 Implement a "round-trip" pipeline in a DCC to and from USD.
4.5 Use USD schemas to support nonstandard attributes/structures during import/export.
4.6 Write a validator for ensuring integrity of an OpenUSD asset exported from a DCC.
4.7 Write an exporter or converter to USD.
4.8 Write or extend a USD importer in a DCC.

NVIDIA Courses and Suggested Readings

Course Reference

> Learn OpenUSD: Developing Data Exchange Pipelines

Tutorials

> Traversing a Stage
> Converting Between Layer Formats

Glossary

> Over
> Stage Traversal

Technical References

> Conceptual Data Mapping
> USD Toolset - usdchecker
> USD Toolset - usdcat
> USD Toolset - usdzip
> USD FAQ - What File Format Is My .usd File?
> USDZ File Format Specification
> Maximizing USD Performance

API Docs

> Encoding Stage UpAxis
> Creating New Schema Classes With usdGenSchema
> Common Idioms and Examples
> UsdGeomPoints - Detailed Description
> UsdProperty::GetNamespace()
> UsdGeomModelAPI::GetExtentsHint()
> UsdGeomBoundable::ComputeExtentFromPlugins()
> UsdGeomBoundable::GetExtentAttr()

Data Modeling: Exam Weight 13%

Require understanding of USD and Sdf data structures and data types, including prims, properties (attributes/relationships), primvars, valueTypes (float, token, matrix4d, etc.), timeSamples, and built-in USD schemas.
5.1 Add a primvar to a mesh.
5.2 Choose the appropriate value types to store attribute data.
5.3 Represent custom metadata.
5.4 Retrieve properties of a prim.
5.5 Understand what causes unexpected visual results.
5.6 Update the extent attribute of a mesh after having updated its points.

NVIDIA Courses and Suggested Readings

Course Reference

- Learn OpenUSD: Setting the Stage
- Learn OpenUSD: Scene Description Blueprints
- Learn OpenUSD: Beyond the Basics

Tutorials

- Transformations, Time-Sampled Animation, and Layer Offsets
- Generating New Schema Classes

User Guides

- Rendering User Guide - Working With Primvars

Glossary

- TimeCode
- Layer Offset
- List Editing
- Primvar

Technical References

- USD Proposals - Spline Animation in USD
- Maximizing USD Performance

API Docs

- Basic Datatypes for Scene Description Provided by Sdf
- Encoding Stage Linear Units
- UsdAttribute - Attribute Interpolation
- UsdGeomPrimvar - Detailed Description
- UsdGeomPrimvar - Interpolation of Geometric Primitive Variables
- Sdf: Scene Description Foundations - Types
- Creating New Schema Classes With usdGenSchema
- UsdGeomBoundable
- UsdGeomXformable
- UsdGeomXformCache
- UsdStage::SetInterpolationType()

Debugging and Troubleshooting: Exam Weight 11%

Introspecting USD stages for the purpose of fixing unexpected or undesired composition results, identifying bad authored data, or optimizing load and render times.
6.1 Identify when using SdfChangeBlocks can alleviate performance bottlenecks.
6.2 Identify why opinions from one layer do not take effect in a composed stage.
6.3 Resolve issues related to asset management.
6.4 Understand what causes unexpected visual results.
6.5 Work with diagnostics for debugging and profiling (TfDebug, diagnostic delegates, Trace, TfMallocTag, ...).

NVIDIA Courses and Suggested Readings

Course Reference

- Learn OpenUSD: Creating Composition Arcs

Tutorials

- Inspecting and Authoring Properties
- Traversing a Stage

Glossary

- LIVERPS
- Reference
- Visibility
- IsA Schema
- Inherits

Technical References

- Maximizing USD Performance
- USD Toolset - usdcat
- USD FAQ - What's the Difference Between an "Over" and a "Typeless Def"?
- usdview LayerStack tab

API Docs

- UsdPrim
- UsdReference - Expressing References Without Prim Paths
- UsdGeomModelAPI - Draw Modes
- UsdProperty::GetPropertyStack()
- UsdUtilsCoalescingDiagnosticDelegate
- TfDiagnosticMgr::Delegate
- UsdStage::Open() - pathResolverContext
- UsdStage - Variant Management
- UsdStage - Working Set Management
- UsdStage::MuteLayer()
- UsdReferences - Reasons Why Adding a Reference May Fail...
- Pcp: PrimCache Population (Composition) - Errors

Pipeline Development: Exam Weight 14%

High-level tasks that are important for a well-rounded OpenUSD developer or architect, including designing the pipeline, asset management, versioning, diagraming, documenting, UI/UX, writing a USD exporter hook to transform data into your pipeline's preferred structure, managing build configurations, or flattening and removing proprietary dependencies from an asset.

7.1	Convert OpenUSD assets to common 3D formats (such as glTF), and ensure fidelity.
7.2	Document asset structure guidelines.
7.3	Explain trade-offs between USDC and USDA formats (performance, readability, archival, etc.).
7.4	Implement "round-trip" pipelines between a DCC and USD.
7.5	Integrate custom resolvers to manage asset paths dynamically.
7.6	Represent custom metadata.
7.7	Validate asset paths are formatted correctly.
7.8	Write or extend a USD importer in a DCC.

NVIDIA Courses and Suggested Readings

Course Reference

- Learn OpenUSD: Setting the Stage
- Learn OpenUSD: Scene Description Blueprints
- Learn OpenUSD: Composition Basics
- Learn OpenUSD: Beyond the Basics
- Learn OpenUSD: Creating Composition Arcs
- Learn OpenUSD: Asset Structure Principles and Content Aggregation

Tutorials

- Authoring Variants

Glossary

- Subcomponent
- Crate File Format
- TimeCode
- Flatten
- Edit Target

Technical References

- USD FAQ - I Have Some Layers I Want to Combine: Should I Use SubLayers or References?
- Principles of Scalable Asset Structure in OpenUSD
- ASWF Asset Structure Guidelines

API Docs

- UsdStage::Flatten()
- Edit Target - Detailed Description
- UsdStage::Traverse()
- UsdPhysics: USD Physics Schema
- Encoding Stage Linear Units
- Encoding Stage UpAxis
- SdfChangeBlock
- UsdNotice
- UsdGeom: USD Geometry Schema - Applying Timesampled Velocities to Geometry

Visualization: Exam Weight 8%

Tasks related to UsdGeom, UsdShade, and UsdLux USD domains (e.g. meshes, cameras, materials, and lights). These are domains that are used in almost every USD use case so we would expect a developer to be more familiar with these domains.
8.1 Add a primvar to a mesh.
8.2 Assign UsdPreviewSurface materials for asset visualization.
8.3 Bind materials to a mesh.
8.4 Create a UsdPreviewSurface shading network to read diffuse colors from a primvar.

NVIDIA Courses and Suggested Readings

Course Reference

- Learn OpenUSD: Setting the Stage
- Learn OpenUSD: Scene Description Blueprints
- Learn OpenUSD: Beyond the Basics

User Guides

- Rendering With USD

Glossary

- Gprim

API Docs

- UsdShadeMaterial - Detailed Description
- UsdGeomSubset - Detailed Description
- UsdShadeMaterialBindingAPI - Detailed Description
- UsdGeomMesh - Detailed Description
- UsdLuxLightAPI
- UsdLuxShadowAPI
- UsdLuxDistantLight
- UsdLuxShapingAPI
- UsdGeom: USD Geometry Schema - Imageable Purpose

Certification Sample Questions

This section provides a set of sample questions designed to test your understanding of key OpenUSD concepts across various domains. These questions are representative of the type and difficulty you might encounter in a certification exam, covering areas such as composition, data modeling, pipeline development, Customizing USD, visualization, content aggregation, and data exchange. Reviewing these questions will help you assess your knowledge and prepare for a certification exam effectively.

Question 1

Domain: Composition

Question: What is the primary purpose of the defaultPrim metadata in a USD layer?

Answer Choices:

A. It specifies which prim is used as the root when the layer is referenced.

B. It defines which material is used as the default for geometry in the layer.

C. It sets which timeCode is used as the default for animation in the layer.

D. It specifies the default prim type when defining a new custom IsA schema.

Question 2

Domain: Composition

Question: Given the following USD layer structure, what will be the final composed value of the xformOp:translate attribute on /World/Chair on a stage with root layer “scene.usda”?

chair_base.usda

```
None
#usda 1.0

def Xform “Chair” {
    double3 xformOp:translate = (0, 0, 0)
    uniform token[] xformOpOrder = [“xformOp:translate”]
}
```

chair_repositioned.usda

```
None
#usda 1.0

def Xform “Chair” {
    double3 xformOp:translate = (1, 0, 0)
}
```

scene.usda

```
None
#usda 1.0

def Xform “World” {
    def Xform “Chair” (
        prepend references = [@./chair_base.usda@, @./chair_repositioned@]
    ) {
        double3 xformOp:translate = (0, 1, 0)
    }
}
```

Answer Choices:

A. (0, 0, 0)

B. (1, 0, 0)

C. (0, 1, 0)

D. (1, 1, 0)

Question 3

Domain: Data Modeling

Question: Which of the following attributes are required to properly define a UsdGeomMesh prim?

Answer Choices: (Select three options.)

A. points

B. faceVertexIndices

C. primvars:displayColor

D. extent

E. normals

Question 4

Domain: Data Modeling

Question: You have a UsdGeomMesh with 1,000 vertices and 500 faces. The mesh uses faceVertexIndices to define triangular faces. What should be the length of the faceVertexIndices array?

Answer Choices:

A. 500—one index per face

B. 1,000—one index per vertex

C. 1,500—three indices per triangular face

D. 2,000—four indices per face (including winding order)

Question 5

Domain: Pipeline Development

Question: When creating a procedural mesh in USD, which attributes must be kept synchronized to ensure the mesh remains valid?

Answer Choices: (Select two options.)

A. points and extent—when vertex positions change

B. faceVertexIndices and faceVertexCounts—when topology changes

C. normals and primvars:displayColor—when shading changes

D. purpose and visibility—when rendering properties change

Question 6

Domain: Customizing USD

Question: You want to create a custom USD schema that adds physics properties to geometry prims. Which base class should your schema inherit from?

Answer Choices:

A. UsdSchemaBase

B. UsdTyped

C. UsdAPISchemaBase

D. UsdPhysicsBase

Question 7

Domain: Pipeline Development

Question: What are some primary advantages of using USD's composition system in a production pipeline?

Answer Choices: (Select two options.)

A. Layers enable version control and tracking of asset changes across the pipeline.

B. Layers allow multiple users to collaborate on the same scene without conflicts.

C. Layers support nondestructive editing and make it easy to revert or update changes.

D. Layers improve scene performance by optimizing data organization and access.

Question 8

Domain: Pipeline Development

Question: When designing a USD-based pipeline, which architectural decisions are most critical for long-term success?

Answer Choices: (Select two options.)

A. Establishing clear and consistent asset naming conventions and directory structures.

B. Implementing comprehensive error handling and validation at all pipeline boundaries.

C. Defining an asset structure once that will meet all present and future requirements.

D. Requiring that all content is always authored, managed, and encoded in USD format.

Question 9

Domain: Visualization

Question: What is the primary purpose of the UsdGeomImageable schema in USD?

Answer Choices:

A. It defines common geometric properties for 3D objects.

B. It provides common properties for objects that can be rendered.

C. It manages how materials are assigned to geometric primitives.

D. It defines properties for texture image data for 3D objects.

Question 10

Domain: Visualization

Question: You want to create a material that can be easily customized with different colors and textures. Which USD shading approach would be most appropriate?

Answer Choices:

A. Create a single material with hardcoded shader parameters.

B. Create a material with exposed parameters that can be overridden.

C. Create separate materials for each color/texture combination.

D. Use only the default material and modify it at render time.

Question 11

Domain: Content Aggregation

Question: What happens when you reference a USD file with metersPerUnit = 1.0 (meters) into a stage with metersPerUnit = 0.01 (centimeters)?

Answer Choices:

A. The USD runtime automatically rescales the referenced geometry to match the stage's unit system.

B. The referenced geometry appears 100 times smaller than intended, as USD does not automatically convert units.

C. The referenced geometry appears 100 times larger than intended, due to USD scaling it up automatically.

D. The referenced geometry is not composed onto the stage, because the unit systems do not match.

Question 12

Domain: Data Exchange

Question: The following is a snippet of a USDA layer output by a new digital content creation (DCC) application for an exported asset. What is the error with the material binding in this export?

```
None
#usda 1.0
(
    defaultPrim = “World”
    metersPerUnit = 0.01
    upAxis = “Z”
)

def Xform “World” (
    kind = “component”
)
{
    def Mesh “bolt” (
        prepend apiSchemas = [“MaterialBindingAPI”]
    )
    {
        rel material:binding = </Materials/metal> (
            bindMaterialAs = “weakerThanDescendants”
        )
        # Mesh definition...
    }
}

def Scope “Materials”
{
    def Material “metal” { # Material definition... }
}
```

Answer Choices:

A. The use of bindMaterialAs = “weakerThanDescendants” is not valid in this context.

B. The material binding is applied to the Mesh prim, but it should be applied to a Subset instead.

C. The material binding is targeting a prim that is outside the hierarchy of the defaultPrim.

D. The material binding data should go on the Material prim and target prims to bind to.

Question 13

Domain: Debugging and Troubleshooting

Question: An artist reported that the MyBox asset in the main.usda stage does not change to blue even though they are setting the loftedColor variant set to the "blue" variant in that layer. Refer to the main.usda, MyBox.usda, and contents.usda below. Explain why the box is no longer changing to blue.

main.usda

```
None
#usda 1.0
(
    defaultPrim = "World"
)

def Xform "World"
{
    def Xform "MyBox" (
        prepend references = @./MyBox.usda@
        variants = {
            string loftedColor = "blue"
        }
    )
    {
        over "Cube" (
            variants = {
                string color = "red"
            }
        )
        {
        }
    }
}
```

MyBox.usda

```
None
#usda 1.0
(
    defaultPrim = “MyBox”
)

def Xform “MyBox” (
    prepend payload = @./contents.usda@
    prepend variantSets = “loftedColor”
    variants = {
        string loftedColor = “red”
    }
)
{
    variantSet “loftedColor” = {
        “red” {
            over “Cube” (
                variants = {
                    string color = “red”
                }
            )
            {

            }
        }
        “blue” {
            over “Cube” (
                variants = {
                    string color = “blue”
                }
            )
            {

            }
        }
    }
}
```

contents.usda

```
None
#usda 1.0
(
    defaultPrim = “MyBox”
)

def Xform “MyBox”
{
    def Cube “Cube”
    (
        prepend variantSets = “color”
        variants = {
            string color = “red”
        }
    )
    {
        variantSet “color” = {
            “red” {
                color3f[] primvars:displayColor = [(1, 0, 0)]
            }
            “blue” {
                color3f[] primvars:displayColor = [(0, 0, 1)]
            }
        }
    }
}
```

Answer Choices:

A. The “red” variant selection in contents.usda is the strongest opinion and overrides any other opinions.

B. The direct local opinion for the color variant selection in main.usda is stronger than the opinion of the color variant selection authored by loftedColor.

C. The loftedColor variant set does not set “primvars:displayColor” for the Cube so there is no reason why changing loftedColor would change the Cube’s color.

D. Variant selections cannot be authored in main.usda. Variant selections must be made in MyBox.usda before it is referenced.

Sample Questions Answer Key

1. A

2. C

3. A, B, D

4. C

5. A, B

6. C

7. B, C

8. A, B

9. B

10. B

11. B

12. C

13. B

Questions?

Contact us here.