# Interactive Computer Graphics notes CIS 560

* Adam Mally
* [Cis460@seas.upenn.edu](mailto:Cis460@seas.upenn.edu)
* Schedule meetings through email
* Regular office hours held in Moore 100C
* TA office hours: Moore 100C, time posted on course web site
* Course website: [www.cis.upenn.edu/~cis460/16fa/index.html](http://www.cis.upenn.edu/~cis460/16fa/index.html)
* Course Piazza sign-up link: [www.piazza.com/upenn/fall2016/cis460560](http://www.piazza.com/upenn/fall2016/cis460560)
* We do not use canvas

Node a; // a starts in stack and node also starts in stack

Node\* b = new Node(); // b starts in stack and Node starts in heap

Int& b = \*a; //illegal since b would be null

Int\* a = null; //illegal

**int\* const cptr = a;**

cptr can only point to certain memory address, value of a can be changed.

**const int\* cptr; = int const \*cptr;**

the content pointer points to cannot be changed, but pointer itself is mutable.

**const int & cptr;**

both pointer and what pointer refers to (&) cannot be changed.

float length() const;

without const modifier, const variables cannot call this function.

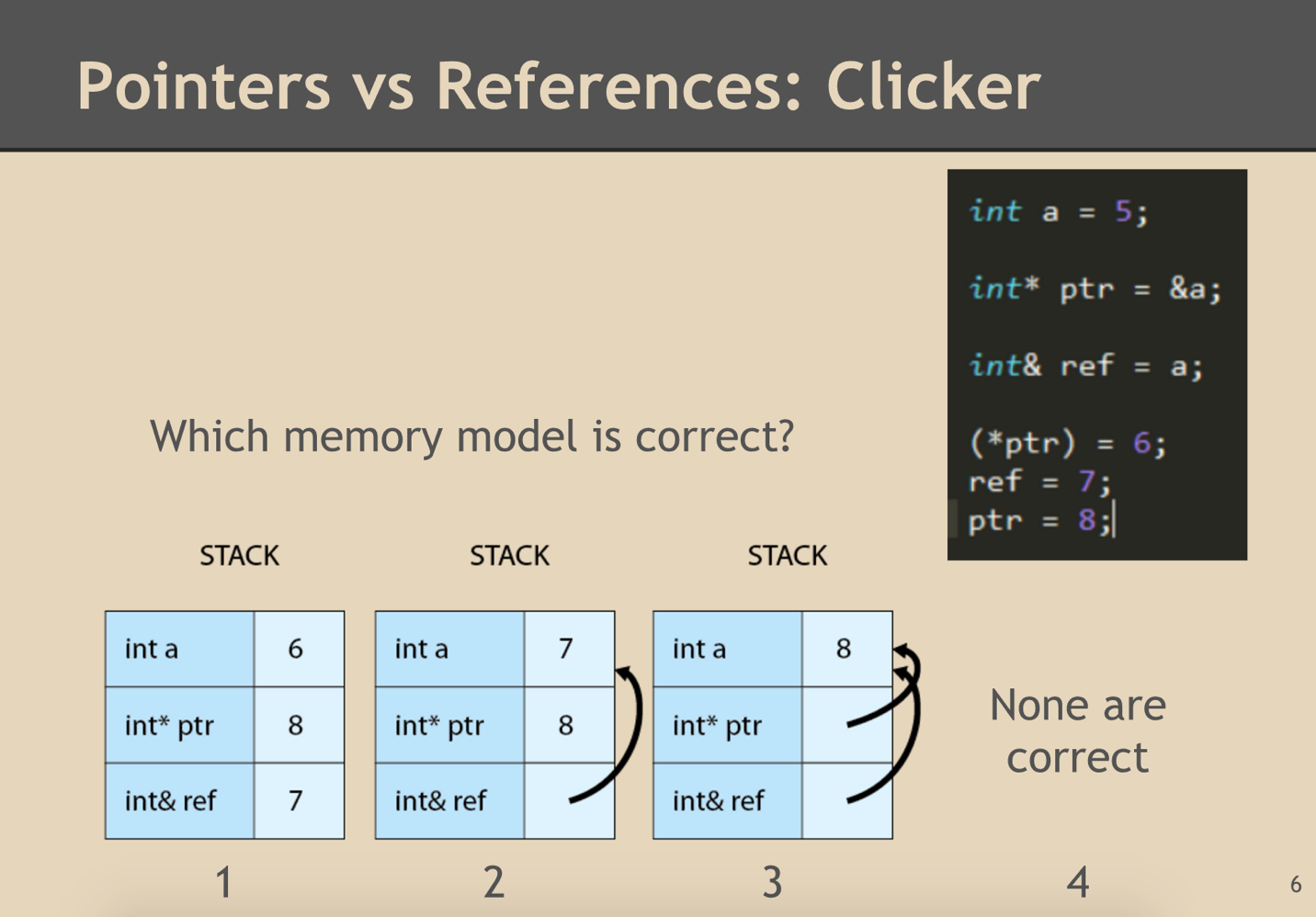
with const modifier, both const and non-const variables can both call length().

you can assign the result of length() to a non-const/const variable.

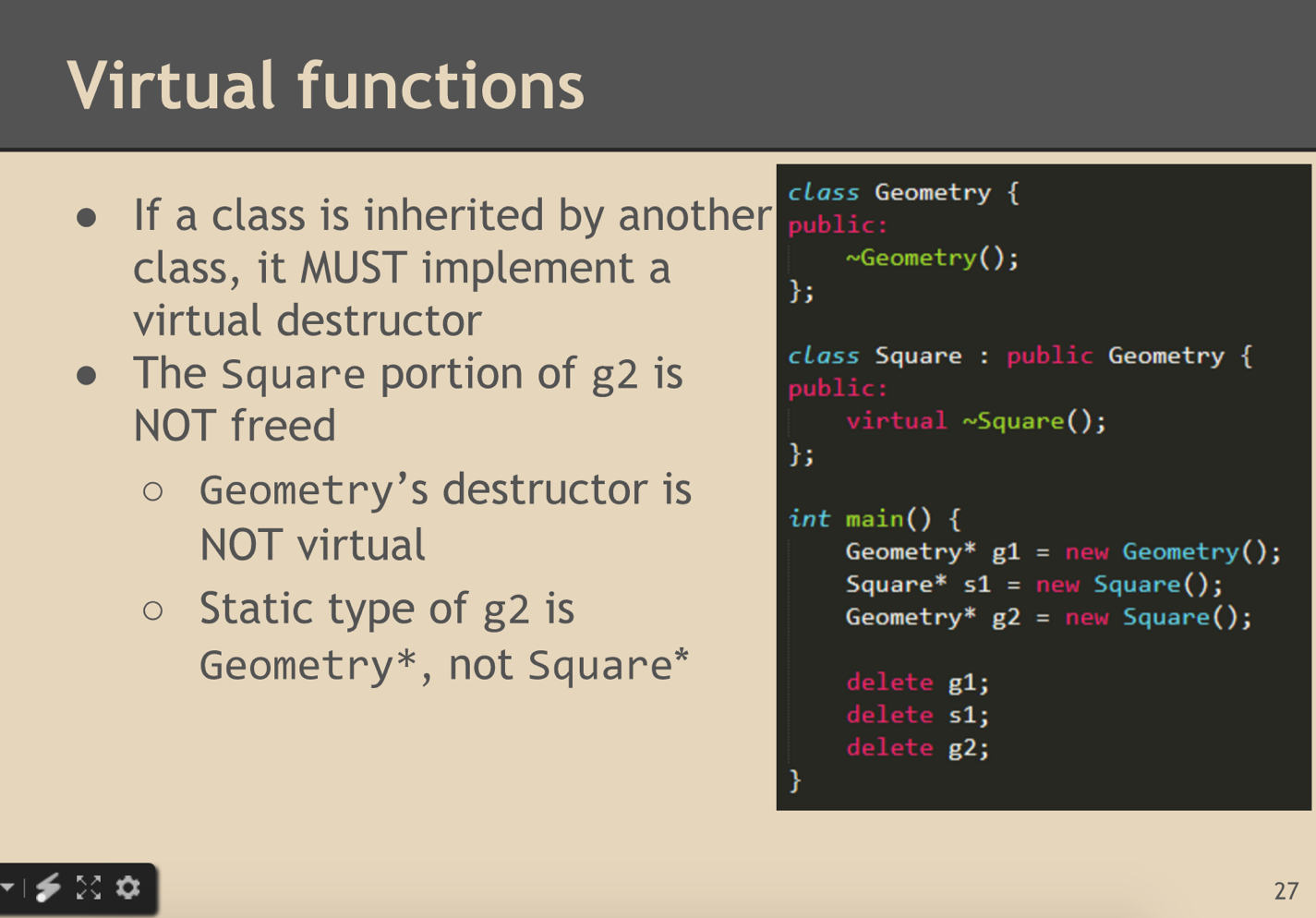
e.c. const float L = v.length(); or float L2 = v.length()

int& GetAgeC() const {} // GetAgeC() is a const function

const int& GetAgeC() {} //GetAgeC() is not a const function, it just returns a const type variable



Answer is 4. Ptr type(int \*) and 8(int) type mismatch, we could use cast to match the type.



Dynamic type of g2 is Square\*.

when you delete a pointer, you should always set the pointer to nullptr.

GL\_ARRAY\_BUFFER; Per-vertex data, e.g. color

GL\_ELEMENT\_ARRAY\_BUFFER; Vertex ordering information

If you declare a variable in a shader file but don’t use this variable, OpenGL will return -1.

Identity transformation matrices

C++ concepts

Back-face culling:

Dot product of the vector(from camera to a point on the triangle) and surface normal, if it’s positive then it’s a back-face.

OpenGL use the order of vertices to determine back-face culling, instead of normal.

Scene graph traversal:

1. Take in a node and transformation matrix, then Recursively call itself.

Void traversal(Node \*n, mat3 T);

// T is not reference, since we only want to use the copy of this matrix instead modifying the original matrix and making it a mess.

// we use post-multiple to concatenate matrices together. T = T\* n->T;

1. If there’s geometry, then draw the geometry with transformation matrix applied.

Shading models:

Shading models dictate how one interpolate the results of reflection models.

Reflection models determine how one determines the color of a surface at a particular point.

OpenGL buffer:

What’s vertex buffer object?

1. Store all information of vertex in the VBO on the GPU
2. Array of floating point numbers

How to read from GPUs:

VIA The IN variables in our shade program.

In each different shader program, what’s the IN variable?

Uniform variable:

Variables that can be read by all threads.

All the same value across all programs.

Read only! IN variables are always read only, OUT variables can be written.

----final----

Transform surface normal:

Multiply the local-space surface normal by the **inverse transpose** of the model matrix to correctly bring it into world space.(Matrix must not include translation components before inverted)

Backward raytracing:

We trace the path of photons in reverse: From each pixel of our camera screen into the scene, and back to the light source.

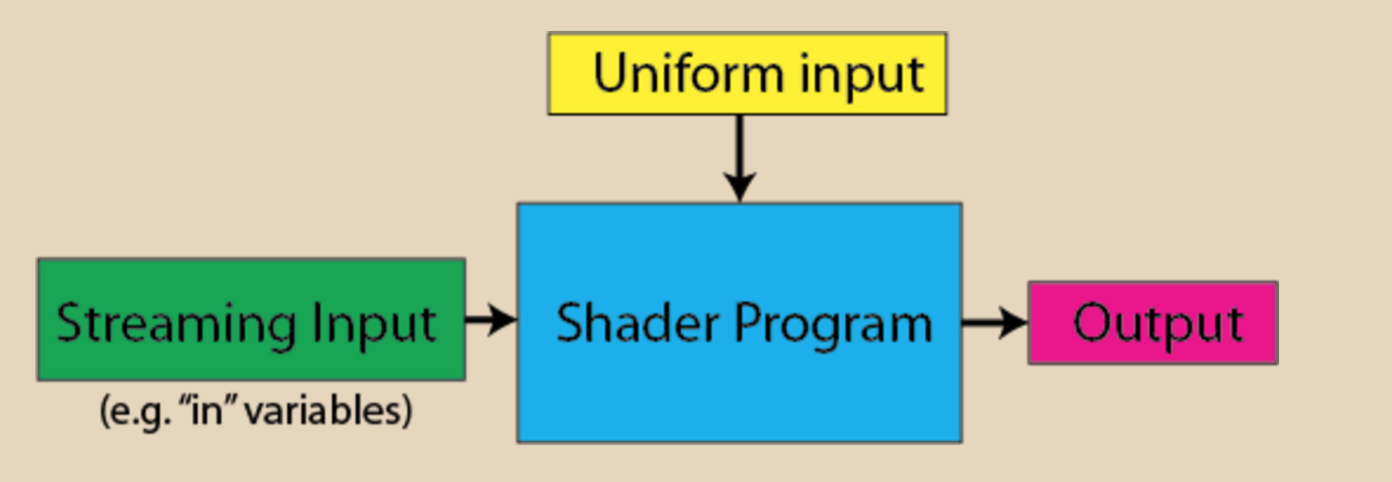
Definition:

Camera rays: emitted thru each pixel of the view screen and are tested against all geometry in the scene. When a camera ray hits a scene object, a light-feeler ray is cast from the point of intersection to each light source in the scene.(reaches its light->contributes a portion of light, otherwise it contributes pure black.)

Catmull-Clark subdivision

1. Initial mesh
2. For each face, compute its centroid
3. Compute the smoothed midpoint of each edge in the mesh
4. Smooth the original vertices
5. For each original face, split that face into N quadrangle faces

Shader Execution Model:



Streaming input is usually things like vertex normal and fragment world positions, uniform inputs are like model matrix, camera matrix, textures, time. Output is to another part of the OpenGL pipeline, such as from a vertex shader to a fragment shader or a fragment shader to a frame buffer.

Vertex shader:

Uniform: model matrix, view matrix

In: vs\_pos, vs\_nor, vs\_col

out: fs\_col, fs\_nor, fs\_pos, fs\_lightvector

Fragment shader:

Uniform: time

In: fs\_pos, fs\_col, fs\_normal, fs\_color

out: out\_col

GLSL:

Like c++ but without:

Pointers, dynamic memory allocation, recursion, user-defined classes

Like c++ with:

Built-in linear algebra types(vector, matrix), built-in constructors, built-in math library, input/output qualifiers

bind matrix: World space to joint local space = inverse of transformation matrix when in bind pose

Transformation matrix \* bind matrix \* vertex pos -> world space position.

Transformation matrix ≠ bind matrix if not in bind pose

**quaternions** are a number system that extends the complex numbers.

Raymarching:

Essentially guess and check your way to a point of intersection

For increasing values of t, generate points along your ray until you find a point that approximately solves your surface equation.(within some tolerance).

Acceleration:

1. Grid: divide space into grid of cubes, each cube stores pointers to elements inside it, test cubes in front-to-back order

Pro: simple, easy to traverse; Con: What size? What if all fall in one cell?

1. Octree: 3-d version of quadtree, divide bounding box into eight equal sub-boxes, recursively subdivide if more than n elements in cell and depth < d; Traverse octree using a stack.

Pro: simple, adapts to geometry, easy to do dynamic updates

Con: handle large objects poorly

1. K-D tree: split along one axis at a time, divide wherever you want

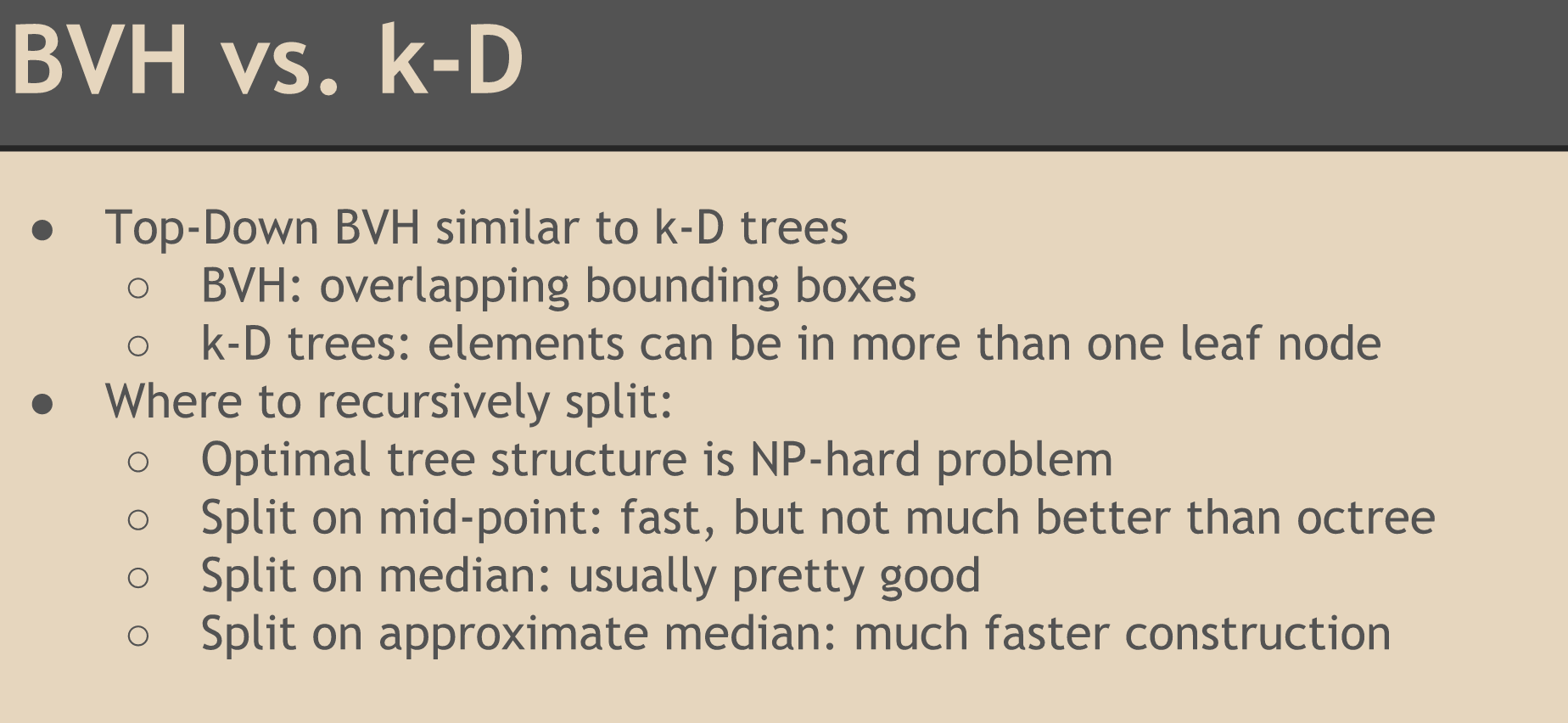
Pro: more flexible, works great for point sets

Con: Extra complexity

1. BSP(Binary Space Partition) trees(better K-D tree): split on arbitrary plane, not just axis-aligned, tighter bounding volumes than k-D tree, still easy to traverse from front-to-back, but really hard to construct.
2. Bounding Volumes:

Pro: tighter bounding boxes than octrees and k-D trees, easy to build, each object in only one left

Con: overlapping boxes, hard to traverse



Perlin noise:

Some parameters:

Frequency: 1, 2, 4, 8, 16, 32 sampling times

Persistence: time parameter for original noise function, the higher persistence, the more noisy it would be

Amplitude = persistence ^ numOctaves

Shadow map: Render 2 images: one from eye, the other one from light source. Compare both maps, if the one from eye has correspondence in the one from light source, other it’s not obstructed, otherwise it would be in shadow.

Basic ray tranced shadows:

1. Shoot ray from eye
2. Ray hits a surface
3. Send shadow ray out to check if the light reaches the point
4. If the shadow ray is obstructed, then the point is in shadow.

Advanced shadow mapping:

Want to fix perspective aliasing: Need more detail near the eye and less detail away from the eye

Want to handle texture resolution limits: 1. Maintain constant texture resolution, independent of scene size 2. Center shadow map around eye 3. Shadow map should not cover areas that are out of view

Spherical UV Mapping:

