# Resumen De preguntas del examen de APA.

## Clipping vs Culling.

Culling es el proceso por el cuál se descarta las partes de los vectores 3d que no vemos en una pantalla 2d.

Clipping es el proceso mediante el cual se seleccionan (y marcan) las partes geométricas que se verán, y también las que no sé verán para que a continuación se pueda llevar a cabo el proceso de Culling.

## Taxonomía de Flynn.

**SISD**: Single Instruction, Single Data Stream Ejemplo: Arquitectura clásica x86.
**SIMD**: Single Intruction, Multiple Data Streams: Arquitecturas vectoriales.
**MISD:** Multiple Instruction, Single Data Stream: Control de vuelo de los transbordadores.
**MIMD**: Multiple Instruction, Multiple Data streams: Arquitecturas distribuidas.

## Rasterización.

La rasterización es el proceso por el cual una imagen descrita en formato gráfico vectorial se convierte a un conjunto de píxeles o puntos para ser desplegados de manera digital.

## Renderización.

Proceso por el cual se procesa una imagen 3D para conseguir un resultado en pixeles y pueda ser visto en el ordenador.

Los procesos de renderización son los siguientes:

La cpu pasa una lista de vértices a la GPU, el procesador de vértices los procesa y los agrupa en primitivas, mediante escaneado se convierten a pixeles agrupados en fragmentos y el procesador de pixeles aplica mapa de textura, dando lugar a los pixeles coloreados.

## Tipos de Memoria.

En función de la latencia de acceso:

**Memoria Global:** Accesible en R/W por todos los hilos y por la CPU

**Memoria Local**: Parte de la memoria privada de cada hilo (memoria virtual).
**Memoria Constante:** Es global de solo lectura, puede cargarse en caché de SM para acelerar las trasnferencias.
**Memoria compartida**: Es de tamaño limitado y accesible en R/W por los hilos de un mismo SM.
**Registro**: Mismo número para todos los hilos, accesible en R/W de forma privada.

## Proceso de compilación de CUDA.

En primer lugar, debe compilarse con NVCC, NVCC es un driver de compilación. Una vez se tenga el código, se pasa por cudafe, que separa el código en C para CPU y el código en C para GPU. El código en c pasa por el compilador locar del host, como gcc y crea el objeto para CPU. El código C para GPU pasa por nvopencc, a PTX, directamente a código objeto o interpretado, se pasa por el OCG y se obtiene el objeto código para CPU, y ambos objetos se juntarán en el ejecutable.

## Comparación entre generaciones.

### Fermi.

La arquitectura Fermi respecto a la g80 tiene novedades, como la implementación de la 3ª generación de los SM, introduce nueva planificación de hilos, permitiendo la ejecución paralela de Kernels. Se implementa soporte para el estándar en números de simple y doble precisión. Se unifica el especio de direcciones de memoria, y mejora el rendimiento de las operaciones atómicas.

Aumenta respecto a la G80, en número de transistores, número de CUDA cores, implementa memoria compartida configurable a 48KB o 16KB, caché de nivel 1 configurable y de nivel 2, permite el trabajo con direcciones de memoria de 64 bits y permite hasta 16 Kernels correr simultáneamente.

Aumenta el número de SFUs. Cada SM tiene dos planificadores de Warps y una memoria en el chip 64KB.

### Kepler.

Lanzada en 2012 mejora considerablemente el rendimiento, pero su foco principal es reducir el consumo de energía. Aparecen los nuevos Streaming Multiprocessor llamados SMX y funcionan a la frecuencia de reloj de los gráficos. Al reducir la frecuenta se disminuye el consumo, pero aumenta el rendimento implementando muchos más CUDA cores que la generación anterior.

Posee 4 Graphics Processing Clusters (GPCs) compuestos por dos SMX cada uno. Utiliza PCI Express 3.0. Incluye una caché de 48K de solo lectura.

## Clasificación de Hwang-Briggs

La clasificación de Hwang-Briggs clasifica los computadores en tres tipos:

**Computadores pipeline**
**Computadores matriciales**
**Sistemas multiprocesador**

# Parte Scala

## Case Class.

Es una construcción que e ayuda cuando se escriben estructuras de datos regulares no encapsuladas. Se añade case delante de la clase que queramos crear, y el compilador de Scala le añade características sintáticas a la clase. En primer lugar, añade un método con el nombre de la clase para poder construir un nuevo objeto con el nombre de la clase. Otra de las características añadidas es que toda la lista de parámetros de la clase implícitamente lleva el prefijo val, por lo que se mantienen como campos.

En tercer lugar, se añade implementaciones naturales como métodos toString, hashCode y equals a la clase. Y por último añade un método copy a la clase para realizar copias modificadas.

```
abstract class Expr
  case class Var(name: String) extends Expr
  case class Number(num: Double) extends Expr
  case class UnOp(operator: String, arg: Expr) extends Expr
  case class BinOp(operator: String,

left: Expr, right: Expr) extends Expr
```

## Pattern Match.

Un pattern match incluye una secuencia de alternativas, cada una empezando por la palabra case. Cada alternativa incluye un patrón y una o más expresiones, que serán evaluadas si el patrón concuerda. El símbolo => separa los patrones de las expresiones. Una expresión match es evaluada intentando que cada patrón en el orden en el que fueron escritos.

```
UnOp("-", UnOp("-", e)) => e // Double negation BinOp("+", e, Number(0)) => e // Adding zero BinOp("*", e,
Number(1)) => e // Multiplying by one
```

## Objetos Singleton.

Por definición son instancias de una única clase. Se utilizan en Scala para reemplazar los miembros de clase estáticos que existen en Java. Se definen sin la utilización de class para ello.

## Traits.

Los traits son una unidad fundamental de reutilización de código en Scala. Un trait encapsula definiciones de métodos y campos, que pueden ser utilizados mezclándolo entre clases.

Ejemplo:

```
 trait Philosophical {
   def philosophize() {
     println("I consume memory, therefore I am!")
   }

}
```

## Mixins.

Un mixin es cuando un trait se mezcla con una clase. Un ejemplo sería

```
      trait SimpleRecipes { // Does not compile
        object FruitSalad extends Recipe(
          "fruit salad",
          List(Apple, Pear),  // Uh oh
          "Mix it all together."

)

        def allRecipes = List(FruitSalad)
      }
```

El orden de los mixins es importante, van de derecha a izquierda. Cuando se llama a un método de una clase con mixins, se llama primero al método trait más a la derecha, pero si este método a su vez llama a un super, en este caso invocará al siguiente trait a la izquierda.

## Jerarquía de clases en Scala.

Tenemos en primer lugar la clase any, de la cuales heredan las clases AnyVal y AnyRef. En AnyVal encontramos clases donde encontramos los elementos básicos de scala, como Int,long,char, y encontramos conversiones implícitas entre ellos. En AnyRef encontramos clases más complejas, tales como List o String.
También tenemos Nothing en escala de la cual tenemos Null que se utiliza cuando aplicamos código de java.

### Rich wrapper.

Son métodos disponibles como con conversión implícita que suele estar junto a los tipos básicos. Como ejemplo encontramos 0 max 5.

### Objetos anónimos.

Son objetos que no se declaran como tal y pueden ser utilizados en cualquier punto del código, un ejemplo sería:

```
val f1 = Foo {

    println("hello from the `f1` instance")

    "this is the result of the block of code"

}
```

## AppEngine.

### Características de Google App Engine:

Ofrece una plataforma completa para el alojamiento y escalado automático de aplicaciones, con servidores de aplicaciones Python y Java, bases de datos BigTable, y un sistema de ficheros GFS. Limita la responsabilidad del programador al desarrollo y primer despliegue, toma control de los picos de tráfico, y es fácilmente integrable con otros servicios de Google. Admite aplicaciones escritas en varios lenguajes, como Java, Python, Ruby, GO, se puede utilizar de manera gratuita y escala fácilmente.

## Tipos de nube en una empresa:

**Nube Privada**: Destinada al uso exclusivo, posee grandes medidas de seguridad, utilizada de forma externa o interna, da unos altos costes.

**Nube Pública:** Es barata, su uso no reside en la propia compañía, posee una infraestructura multiuso. Menos segura que la primera, y menos poder de computo.

**Nube híbrida**: Comprende tanto una nube privada, como una nube pública.

## JSP, JSO, JPA:

**JSP:** Plantillas usadas para diseñar una interfaz de usuario en archivos independientes. Cuando se carga por primera vez, el servidor de desarrollo lo convierte en código Java. Al subirlo a App Engine, el SDK compila todas las JSP en código de Bytes y sube únicamente eso.

**JDO:** Objetos de Datos Java, es un estándar API para el almacén de datos. Al crear clases JDO se deben utilizar anotaciones Java para describir como se deben guardar las instancias en el almacén de datos.

**JPA:** Otro estándar Api para el almacén de datos. Ambos estándares los proporciona DataNucleus Access Platform.

## Características de GWT:

Las aplicaciones GWT pueden tener varias ventanas contenidas bajo un único padre. Es sencillo de usar, reduce costes, no necesita servidor, y si se usa, la mayoría de las computaciones se pueden delegar al cliente, optimiza el ancho de banda.

## Manifiesto de GWT:

GWT debe ayudar a crear código estable, eficiente, y compatible con múltiples navegadores. GWT debe ser amigable con los desarrolladores y en primer lugar debe estar el usuario y luego el desarrollador.

## Definición de las nubes:

**On-demand computing**: Es un modelo en el cual los recursos de computación están disponibles a medida que la necesite el usuario.
**Utility Computing**: se define como el suministro de recursos computacionales, como puede ser el procesamiento y almacenamiento, como un servicio medido similar a las utilidades públicas tradicionales.

**Autonomic computing:** se refiere a las características de autogestión de los recursos informáticos distribuidos, adaptándose a los cambios impredecibles y ocultando la complejidad intrínseca a los operadores y usuarios
**Platform Computing:** Es una plataforma de computación donde se ejecuta un software.
**Edge Computing:** Es un método de optimización de computación en nube. Consiste en tomar el control de aplicaciones informáticas, datos y servicios lejos de algunos nodos centrales (el" núcleo ") para el otro extremo lógico en el servicio.
**Elastic Computing:** Es un concepto de computación en la nube en la cuál se da la facilidad de escalar los recursos como se necesiten dados por el proveedor.
**Grid Computing**: La computación grid (o en malla) es una tecnología que permite utilizar de forma coordinada recursos heterogéneos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado.