파일처리

1. 개요
2. 파일처리의 7할은 메모리, 2할은 데이터 타입 관리, 나머지 1할이 파일 IO로서 파일처리의 실질적인 주제는 메모리 관리
3. 메모리 관리능력은 프로그래머로서 하수와 중수를 가르는 관건(콜럼버스의 달걀)
4. 데이터 타입 관리는 템플릿의 관건. 템플릿을 사용하지 않는다면 ctrl+CV와 하드코딩이 난무
5. 목표 및 과정
6. 기존 속설을 점검하고 현 파일 IO에 대한 기준 확립
7. 메모리에 대한 관리법 토의
8. 데이터 타입 관리법 토의
9. 자동화 인터페이스
10. 메타 데이터 구현
11. 파일 시스템 구현

**[ 만들어 놓으면 재산인게 바로 메모리/파일 시스템 ]**

1. 기존 속설을 점검하고 현 파일 IO에 대한 기준 확립
2. 파일 IO(하드디스크 드르륵)은 PC관련 처리 중 가장 느린 부분. 그런데 과거에는 CPU와 메모리가 더 느렸음
3. “효과음은 wav로 , 배경음은 mp3 스트리밍으로.” 업계의 당연한듯한 솔루션. WHY?

효과음의 경우 10M 파일 읽는 비용 100. 그대로 읽어서 뿌리면 **100.** 인코딩하면 3M로 줄일 수 있으므로 파일 읽는 비용은 30이 되지만, 디코딩 비용이 100 추가됨. 따라서 최종 비용은 **130**. 결과적으로 파일 크기가 크지 않다면 인코딩/디코딩 과정을 생략하고 원본 그대로 읽어 뿌리는게 유리

* 효과음은 wav로!

배경음의 경우 CD급 음질(44000MHz) 4분 길이의 곡 용량은 평균 50M. 음질을 반으로 줄이고 인코딩하면 5M 정도로 줄일 수 있음. 용량차이가 너무 나기 때문에 인코딩/디코딩 과정을 거치는 대신 스트리밍으로 뿌려 유저가 로딩 및 디코딩 과정을 인지하지 못하도록 숨김

* 배경음은 mp3 스트리밍으로!

1. 동일한 솔루션이 파일 관련된 전반적인 부분에 모두 적용되었음
2. 하지만 현재는 하드디스크를 대체 할 SSD가 나왔고 CPU와 메모리도 비약적으로 빨라짐

하드디스크 IO는 여전히 느리지만 인코딩/디코딩 비용이 급락하여 파일 IO를 최대한 줄이는게 비용 감소에 유리. 따라서 현재는 모두 인코딩이나 압축을 거쳐 파일로 적재하는 편이 이상적이며, 반대급부로 메모리 관리 난이도는 더욱 올라감

1. 메모리에 대한 관리법 토의
2. 관점 하나, 메모리는 데이터 타입의 집합이다

Ex>

typedef struct \_DataSet {

int hp;

std::string name;

float attackPower;

} DataSet;

선언시 형태와 크기가 고정됨. 직관적. 구현 난이도 쉬움

Ex> 0\_StandardMemoryOp

1. 관점 둘, 메모리는 단지 byte array일 뿐이다

Ex> unsigned char\* memory = new unsigned char[2048];

끗.

Ex> 1\_ChunkMemoryOp

1. 보기엔 별거 아닌 말장난처럼 보이지만, 둘 사이에는 콜럼버스의 달걀과도 같은 차이가 존재

* 메타데이터 : 어떤 데이터가 적재될지 며느리도 모름. 오직 런 타임때만 알 수 있음
* Direct3D의 FVF : case by case가 너무 많아서 사전 준비를 할 수 없음

1. 그래서 필요한 것의 메모리의 모듈화와 IO 인터페이스

Ex> 2\_MemoryInterfaceOp

**[ 샘플은 사랑입니다 ^^]**