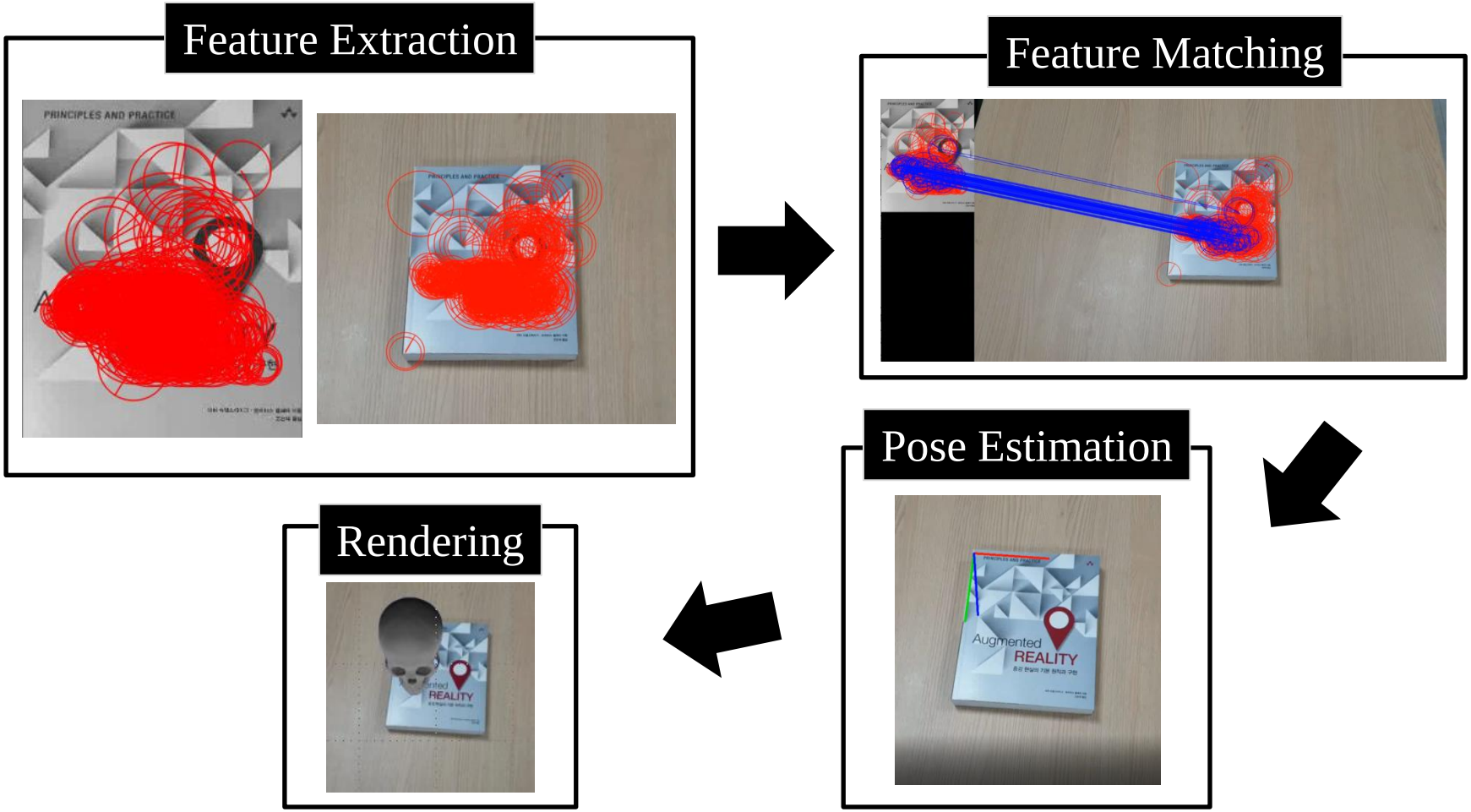
Natural Feature Tracking AR Project

Hyeonah Choi Hyung Jun Cho

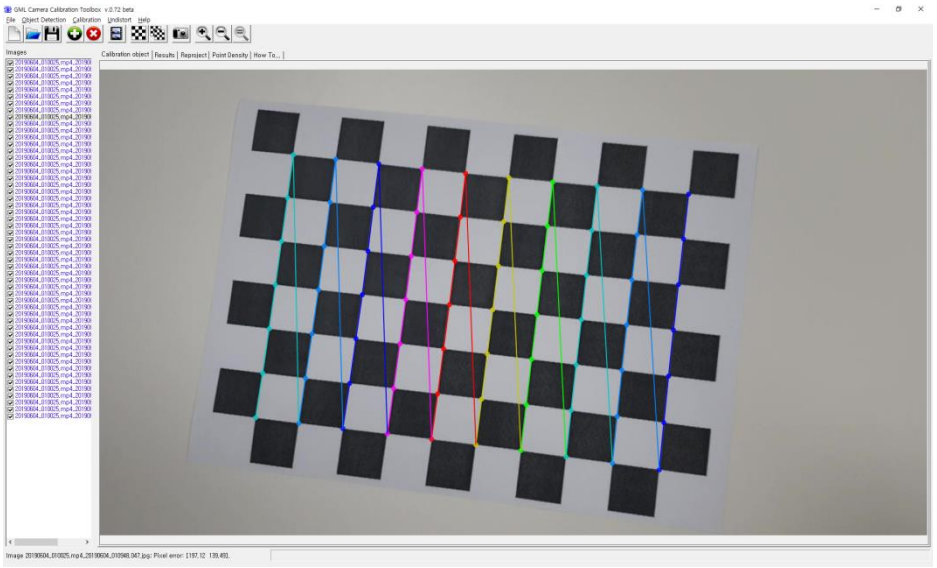
Overview & Setup

- Implemented with C++
- Developed with OpenCV4.1.0, Visual Studio 2017 and DirectX11



Camera Calibration

- Calculate camera intrinsic parameters.
- GML Camera Calibration Toolbox



Parameter	Value
Calibration date	2019-06-04 오후 9:45:49
Number of images	53
Square size	25,000 (mm)
Focal length	[1428,683 1430,750]
Principal point	[644,834 348,817]
Distortion	[0,380035 -1,643414 -0,003960 -0,005340]
The camera matrix	[1428,683 0 644,834; 0 1430,750 348,817; 0 0 1]
Pixel error	[224,87 137,96]

Camera Calibration

- Define projection matrix using given camera parameters.

```
m_fx = 1428.683; m_cx = 644.834;
m_fy = 1730.750; m_cy = 348.817;
m_k1 = 0.380035, m_k2 = -1.643414;
m_p1 = -0.003960; m_p2 = -0.005340;

m_cameraIntrinsicData[0] = m_fx;
m_cameraIntrinsicData[1] = 0;
m_cameraIntrinsicData[2] = m_cx;
m_cameraIntrinsicData[3] = 0;
m_cameraIntrinsicData[4] = m_fy;
m_cameraIntrinsicData[5] = m_cy;
m_cameraIntrinsicData[6] = 0;
m_cameraIntrinsicData[7] = 0;
m_cameraIntrinsicData[8] = 1;

m_cameraDistortionCoefficientData[0] = m_k1;
m_cameraDistortionCoefficientData[1] = m_k2;
m_cameraDistortionCoefficientData[2] = m_p1;
m_cameraDistortionCoefficientData[3] = m_p2;
```

(1) Feature Extraction

- ORB, BRISK, SIFT and SURF Algorithm
  - Create video and transform the video image to grayscale.
  - Select the algorithm.

```
enum MATCH_TYPE
{
  ORB, BRISK, SURF, SIFT
};

MATCH_TYPE matchType = MATCH_TYPE::SIFT;
```

```
switch (matchType)
{
case MATCH_TYPE::ORB:
  feature2dPtr = ORB::create();
  break;

case MATCH_TYPE::BRISK:
  feature2dPtr = BRISK::create();
  break;

case MATCH_TYPE::SURF:
  feature2dPtr = SURF::create();
  break;

case MATCH_TYPE::SIFT:
  feature2dPtr = SIFT::create();
  break;

default:
  feature2dPtr = ORB::create();
  break;
}
```

(1) Feature Extraction

- ORB, BRISK, SIFT and SURF Algorithm
  - Detect feature points and compute descriptors of target image.
  - Draw keypoints to the target image and video image.

```
// Feature Extraction

Mat imgFeatureExtractionResult;
Mat videoImgGray;
cvtColor(m_videoMat, videoImgGray, COLOR_BGR2GRAY);
m_feature2dPtr->detectAndCompute(videoImgGray, Mat(), m_videoKeyPoints, m_videoDescriptors);
drawKeypoints(m_videoMat, m_videoKeyPoints, imgFeatureExtractionResult, Scalar(0, 0, 255),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
```

(1) Feature Extraction Result

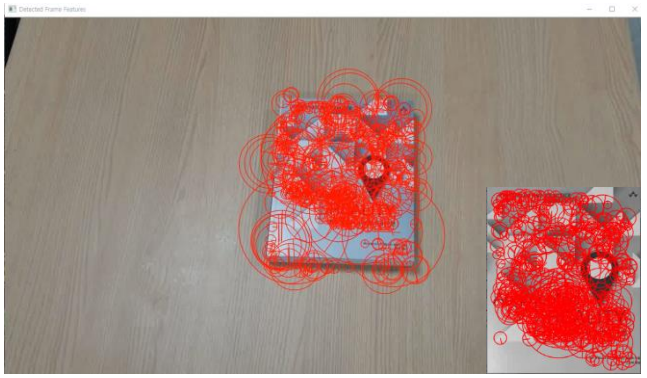
- Comparison - ORB, BRISK, SIFT and SURF.

ORB

BRISK

SIFT

SURF



(2) Feature Matching

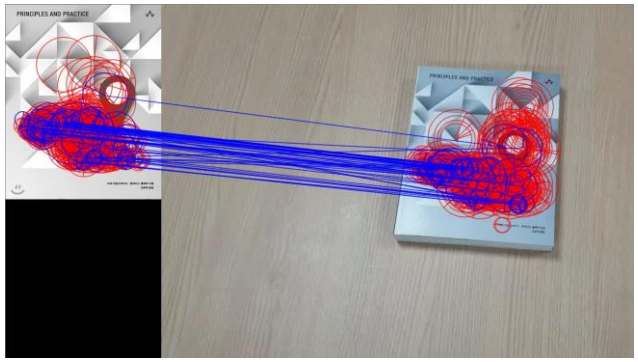
- ORB, BRISK, SIFT and SURF matching
  - Create BF matcher
  - Sort the resulting descriptors using std::sort.
  - Leave the top 10% and erase the rest
  - Draw drawMatches

```
using cv::BFMatcher;
Mat imgFeatureMatching;
m_matcher.match(m_markerDesciptors, m_videoDescriptors, m_matchResult, Mat());
sort(m_matchResult.begin(), m_matchResult.end());
const int goodMatches = (int)(m_matchResult.size() * 0.1f);
m_matchResult.erase(m_matchResult.begin() + goodMatches, m_matchResult.end());
drawMatches(m_markerImage, m_makerKeyPoints, m_videoMat, m_videoKeyPoints,
m_matchResult, imgFeatureMatching, Scalar(255, 0, 0), Scalar(0, 0, 255), Mat(),
DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
```

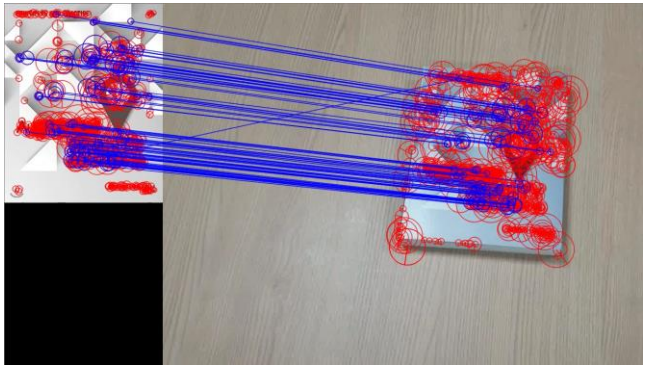
(2) Feature Matching Result

- Comparison - ORB, BRISK, SIFT and SURF.

ORB

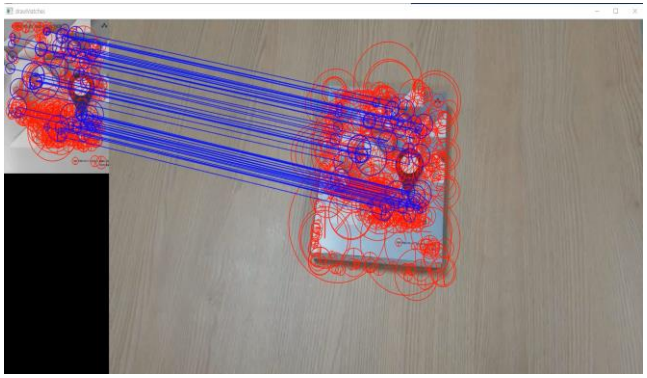


BRISK



SIFT

SURF



(3) Pose Estimation

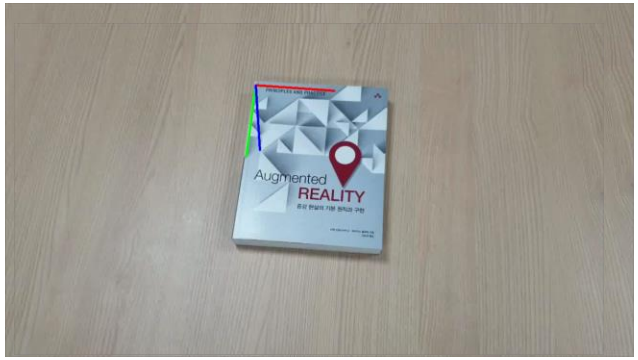
- PnP algorithm
  - Calculate camera pose using solvePnPRansac()
  - Use DrawFrameAxes to visualize the rvec, tvec
  - Change length(200.0f) and thickness(3).

```
Mat imgPoseEstimation;
m_videoMat.copyTo(imgPoseEstimation);

vector<Point3f> objectPoints;
vector<Point2f> imagePoints;
for (const auto& dmatch : m_matchResult)
{
  imagePoints.emplace_back(m_videoKeyPoints[dmatch.trainIdx].pt);
  objectPoints.emplace_back(Point3f(
    m_makerKeyPoints[dmatch.queryIdx].pt.x,
    m_makerKeyPoints[dmatch.queryIdx].pt.y,
    0.0f));
}

Mat rotationVec, positionVec;
solvePnPRansac(objectPoints, imagePoints, m_cameraIntrinsic, m_cameraDistortionCoefficient,
rotationVec, positionVec);
drawFrameAxes(imgPoseEstimation, m_cameraIntrinsic, m_cameraDistortionCoefficient, rotationVec,
positionVec, 200.0f, 3);
```

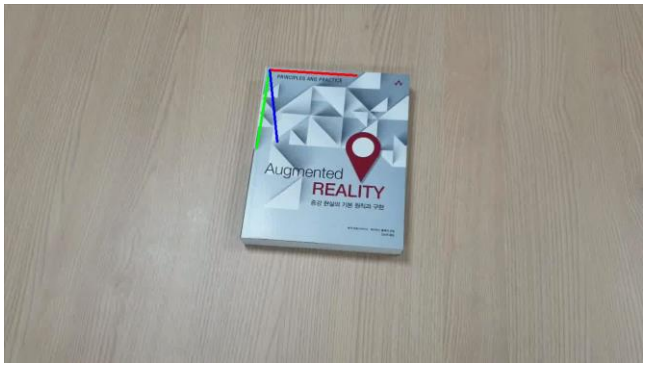
(3) Pose Estimation result

- Comparison - ORB, BRISK, SIFT and SURF.

ORB

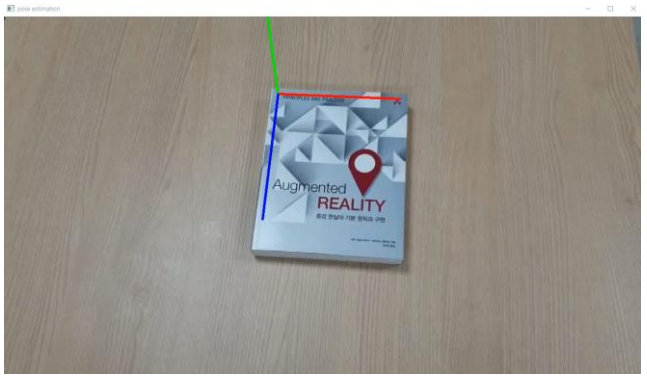


BRISK



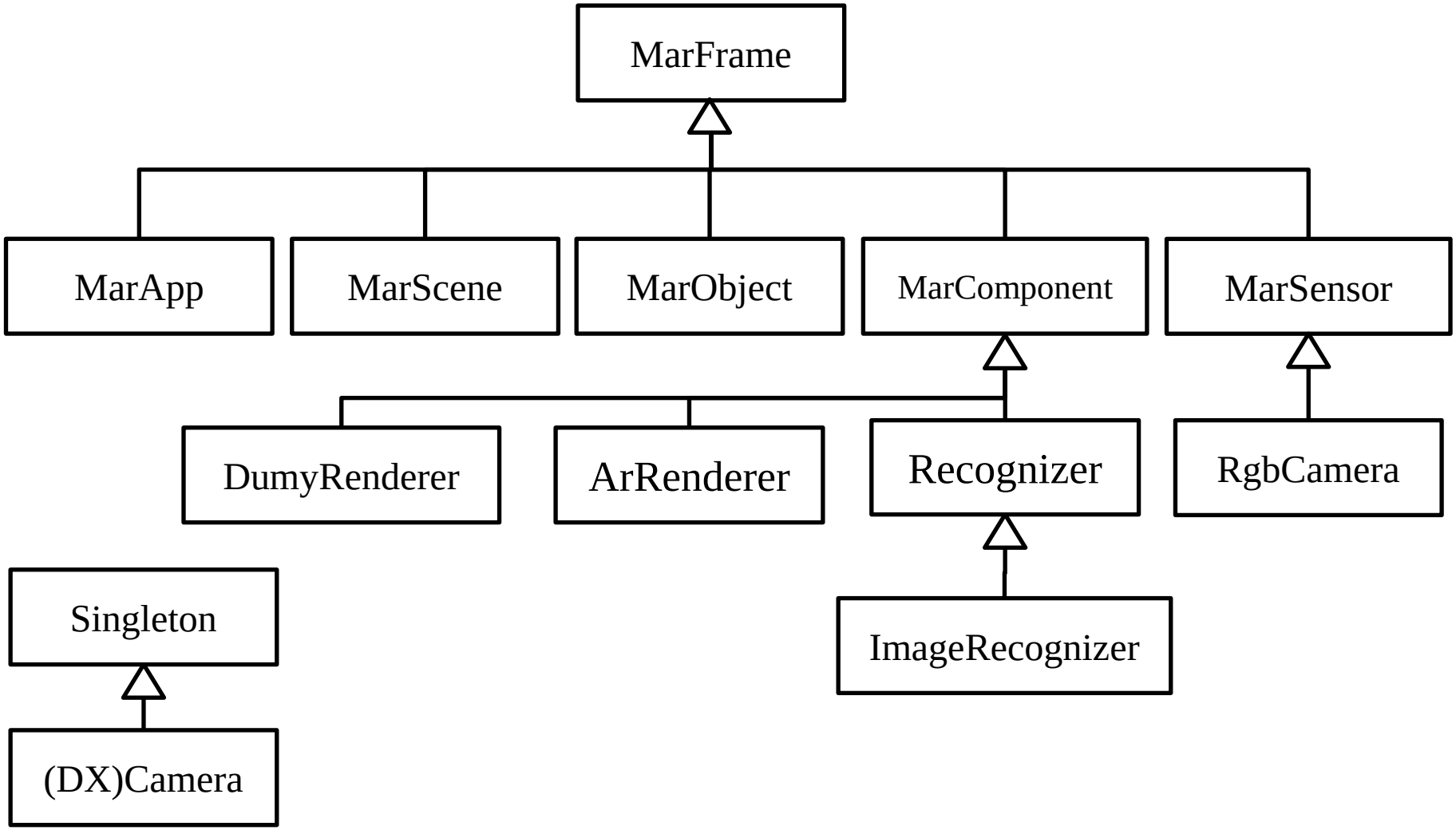
SIFT

SURF



(4) Rendering (DirectX11)

- Class diagram

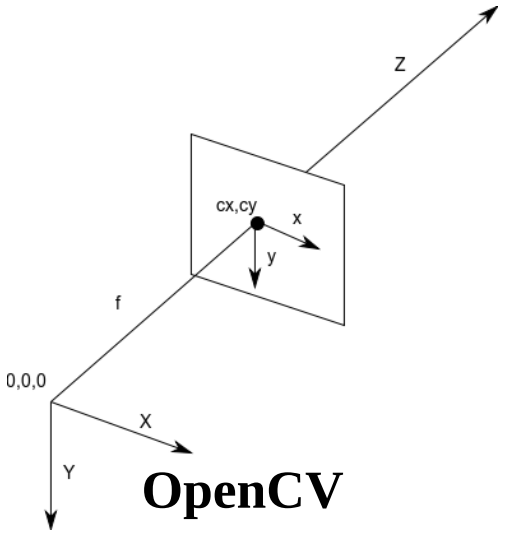


(4) Rendering (DirectX11)

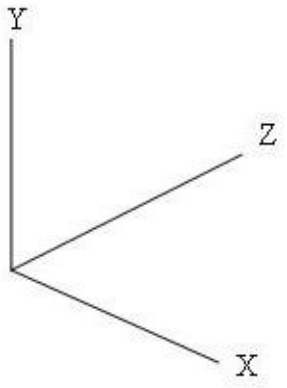
- **DumyRenderer** renders the skull.
- **ImageRecognizer** do Feature Extraction, Feature Matching, Pose Estimation.
- **ArRenderer** captures the DirectX window,
  chroma-processes the blue color,
  and combines it with the video output using OpenCV.

```
VOID MarApp::_Init_()
{
    Effects::InitAll(Global::g_d3dDevice);

    MarScene* scene1 = new MarScene("Scene1", USER_DEFINED_INIT
    {
        GetScene()->AddSensor(new RgbCamera("RgbCamera", "01_Asset/video.avi", false));

        MarObject* object1 = new MarObject("Object1");
        object1->AddComponent<DumyRenderer>();
        object1->AddComponent<ImageRecognizer>();
        object1->GetComponent<ImageRecognizer>()->SetTargetImageAndMatchType("01_Asset/marker.jpg", MATCH_TYPE::BRISK);
        GetScene()->AddObject(object1);

        MarObject* arRenderObject = new MarObject("arRenderObject");
        arRenderObject->AddComponent<ArRenderer>();
        GetScene()->AddObject(arRenderObject);
    });

    SetNextScene(scene1);
}
```

(4) Rendering (DirectX11)



OpenCV



DirectX

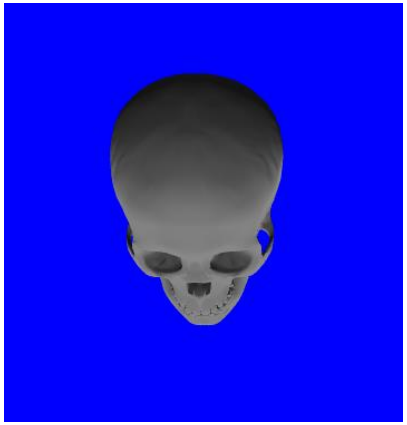
- The coordinate system of OpenCV and DirectX differ only in the direction of the Y axis.
- It changes the coordinate values from OpenCV to apply to DirectX.

(4) Rendering (DirectX11)

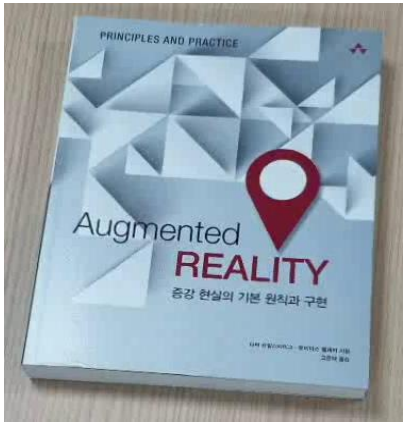
- Because the Y-axis is opposite, we apply - when applying the position.
- Because the Y-axis is opposite, we apply - when applying the rotation.

```
Mat rotationVec, positionVec;
solvePnPRansac(objectPoints, imagePoints, m_cameraIntrinsic, m_cameraDistortionCoefficient, rotationVec, positionVec);

float cameraPosX = (float)(positionVec.at<double>(0, 0)) * 0.005f;
float cameraPosY = (float)(positionVec.at<double>(1, 0)) * 0.005f;
float cameraPosZ = (float)(positionVec.at<double>(2, 0)) * 0.005f;

float cameraRotX = (float)(rotationVec.at<double>(0, 0));
float cameraRotY = (float)(rotationVec.at<double>(1, 0));
float cameraRotZ = (float)(rotationVec.at<double>(2, 0));

float thelta = sqrt(cameraRotX * cameraRotX + cameraRotY * cameraRotY + cameraRotZ * cameraRotZ) * (180 / 3.141592f);
cameraRotX /= thelta;
cameraRotY /= thelta;
cameraRotZ /= thelta;

Camera::GetInstance()->SetPosition(cameraPosX, -cameraPosY, cameraPosZ);
Camera::GetInstance()->SetRotation(cameraRotX, -cameraRotY, cameraRotZ);
```

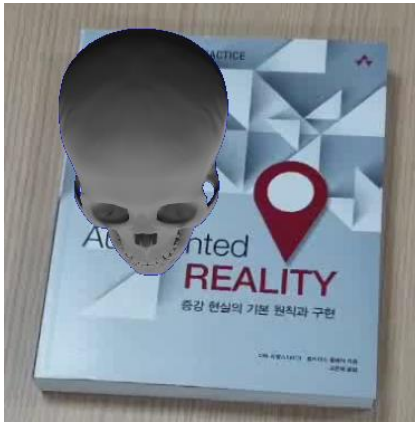
(4) Rendering (DirectX11)



DirectX



OpenCV



Result

- Sets the background color of DirectX to blue and removes the specified color when merging with OpenCV.

(4) Rendering (DirectX11)

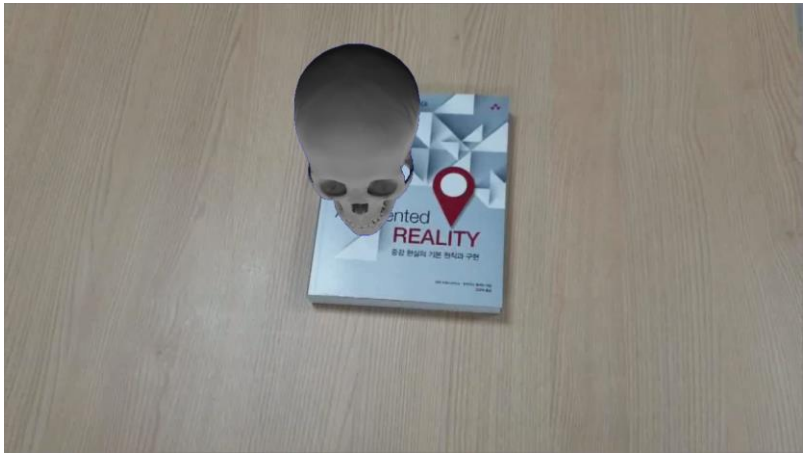
- Capture the screen of DirectX and make the screen size of OpenCV equal to the size of DirectX.
- In the captured DirectX screen, pixels excluding the blue value are combined with the screen of OpenCV to create the final frame.

```
m_matCv = m_marSensor->GetCurrentCaptureFrame();
if (m_matCv.empty())
    return;

m_matDx = hwnd2mat(Global::g_hwnd);

cv::resize(m_matCv, m_matCv, cv::Size(m_matDx.cols, m_matDx.rows));
cvtColor(m_matDx, m_matDx, COLOR_BGRA2BGR);
m_matCv.copyTo(m_matResult);

auto iteratorDxEnd = m_matDx.end<Vec3b>();
auto iteratorResult = m_matResult.begin<Vec3b>();

for (auto iteratorDx = m_matDx.begin<Vec3b>(); iteratorDx != iteratorDxEnd; iteratorDx++, iteratorResult++)
{
    Vec3b bgraDx = (*iteratorDx);
    if (bgraDx[0] != 255 || bgraDx[1] != 0 || bgraDx[2] != 0)
    {
        (*iteratorResult)[0] = bgraDx[0];
        (*iteratorResult)[1] = bgraDx[1];
        (*iteratorResult)[2] = bgraDx[2];
    }
}
```

(4) Rendering result

- ORB, BRISK

ORB



BRISK